

IEEE

MICRO

Chips, Systems, Software, and Applications

DECEMBER 1991

DATABASE MACHINES

*Companion Issue
to December 1991
Computer on
Heterogeneous
Distributed
Database Systems*

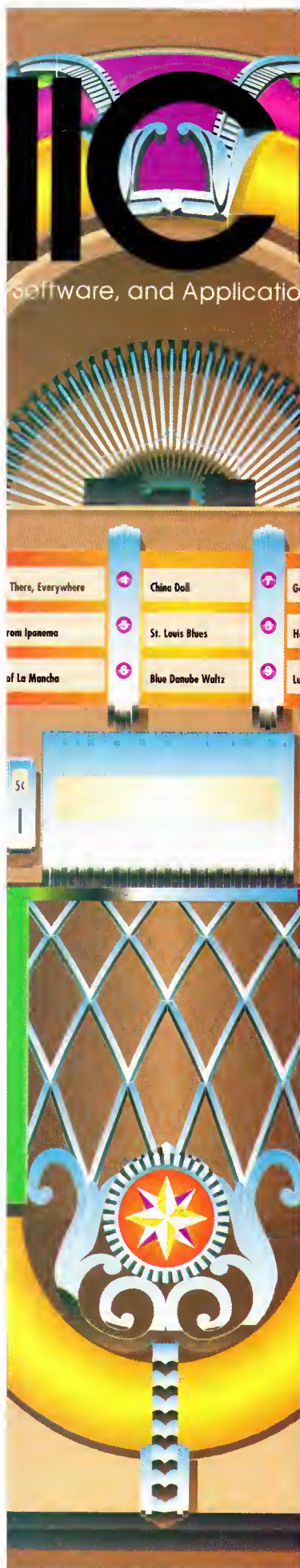
1961-1991



40 YEARS OF SERVICE
IEEE COMPUTER SOCIETY



THE INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS, INC.



IEEE MICRO

Published by the IEEE Computer Society

December 1991

F E A T U R E S

**6 Guest Editors' Introduction:
Database Machines—Trends and
Opportunities**

M. Abdelguerfi and A.K. Sood

**8 VLSI Accelerators for Large Database
Systems**

*Kuo Chu Lee, Takako Matoba Hickey,
Victor W. Mak, and Gary E. Herman*
Resolving the problem of slow response
times in a cost-effective manner

**22 An Associative Accelerator for Large
Databases**

Pascal Faudemay and Mongia Mhiri
Implementing relational operations at
speeds adaptable to advanced micropro-
cessors

**35 A Fine-Grain Architecture for
Relational Database Aggregation
Operations**

M. Abdelguerfi and A.K. Sood
Designing and simulating a prototype
four-input unit for fabrication using
discrete components

**44 A Parallel, Scalable, Microprocessor-
Based Database Computer for
Performance Gains and Capacity
Growth**

David K. Hsiao

Using a variable number of processors to
produce an experimental computer

**61 Rinda: A Relational Database
Processor with Hardware Specialized
for Searching and Sorting**

*Ushio Inoue, Tetsuji Satoh, Haruo Hayami,
Hideaki Takeda, Toshio Nakamura, and
Hideki Fukuoka*

Reducing a host computer's CPU and I/O
times with specialized hardware

**71 Special Feature
Annual Index—Volume 11**

Cover design: Alexander Torres

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: \$21 in addition to IEEE Computer Society or any other IEEE society member dues; \$38 for members of other technical organizations. This journal is also available in microfiche form.

Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices.

Copyright and reprint permissions: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US Copyright Law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress St., Salem, MA 01970. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to Permissions Editor, *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Copyright © 1991 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

DEPARTMENTS

- 2 Letters/Mailbag**
On an open architecture
- 4 Micro News**
PLA copyright infringement
- 78 Micro Law**
Database system copyrights
- 80 On the Edge**
SBus: An open architecture
- 84 Micro Standards**
Computing interconnections
- 86 New Products**
Workstations; boards and cards;
communications; chips and
components

CS membership application, p. 21; Reader Interest/Service/Subscription cards, p. 64A; Advertiser/Product Index, cover 3; CS information page, cover 4

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821-8380

Editorial: Unless otherwise stated, bylined articles and descriptions of products and services reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society. Send editorial correspondence to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. All submissions are subject to editing for style, clarity, and space considerations.

EDITOR-IN-CHIEF

Dante Del Corso
*Politecnico di Torino**

ASSOCIATE EDITOR-IN-CHIEF

Ashis Khan
*Mips Computer Systems, Inc.***

EDITORIAL BOARD

John Crawford
Intel Corporation

K.E. Grosspietsch
GMD

Joe Hootman
University of North Dakota

David K. Kahaner
National Institute of Standards and Technology

Hubert D. Kirmann
Asea Brown Boveri Research Center

Priscilla Lu
AT&T

Richard Mateosian

Nadine E. Miner
Sandia National Laboratories

Ken Sakamura
University of Tokyo

John L. Schmalzel
University of Texas at San Antonio

Michael Slater
Microprocessor Report

John W. Steadman
University of Wyoming

Richard H. Stern

Philip Treleaven
University College London

Carl Warren
McDonnell Douglas Space Systems Co.

Maurice Yunik
University of Manitoba

MAGAZINE ADVISORY COMMITTEE

James J. Farrell, III (chair)
Fiorenza Albert-Howard

Valdis Berzins
Jon T. Butler

B. Chandrasekaran
Carl Chang
Manuel d'Abreu
Dante Del Corso
Gerald Engel
Michael Evangelist
Sushil Jajodia
H.T. Seaborn
Pradipt K. Srimani
Peter R. Wilson

STAFF

Marie English
Managing Editor

David Sims
Assistant Editor

H.T. Seaborn
Publisher

Marilyn Potes
Editorial Director

Douglas Combs
Assistant Publisher

Pat Paulsen
Assistant to the Publisher

Jay Simpson
Art Director

Joseph Daigle
Production

Christina Champion
Membership/Circulation Manager

Heidi Rex,
Marian Tibayan
Advertising Coordinators

PUBLICATIONS BOARD

Ronald G. Hoelzeinan (chair)

James Aylor
Victor R. Basili
Richard Burke
Jon T. Butler
J.T. Cain

B. Chandrasekaran
Carl Chang
Manuel d'Abreu
Dante Del Corso
Gerald Engel
Michael Evangelist
James J. Farrell, III
Tse-yun Feng
Anil K. Jain
Ted Lewis
Ray Miller
Michael C. Mulder
C.V. Ramamoorthy
H.T. Seaborn
Sallie Sheppard
Harold Stone
Earl E. Swartzlander
Peter R. Wilson

* Submit six copies of all articles and special-issue proposals to Dante Del Corso,
Dipartimento di Elettronica, Politecnico di Torino,
C.so Duca degli Abruzzi, 24, 10129 Torino, Italy; phone +39 11 564 4044;
Compmail: d.delcorso; Internet: delcorso@polito.it; Bitnet: delcorso@itopoli
or

** Ashis Khan. Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086;
Internet: ashis@mips.com.



On an open architecture

To the Editor:

In the October 1991 issue I particularly liked "A RISC Processor for Embedded Applications Within an ASIC," "Performance and the i860," Micro News, Micro Law, and Software Report.

One criticism: In "On the Edge" about IEEE Std P1754, the P1754 architecture is compared to several commercially available microprocessors in Table 1. This table is preposterously misleading. It is clearly biased toward the P1754 architecture. For example, it claims that there are "many" performance grades of P1754 but "few" of the 68000 architecture, and states that those are "mainly different clock speeds."

The author describes different performance grades as differing in cache implementation, external bus structure, basic technology, etc. Without even looking up references, I can cite the following variants on the 68000 architecture which differ in these respects:

68008—8-bit data bus, 20-bit address bus;

68000—16-bit data bus, 24-bit address bus;

68HC000—CMOS;

68020—256-byte instruction cache, extra address modes, 32-bit address and data buses, dynamic data bus sizing;

68030—separate 256-byte instruction and data cache, MMU;

68040—4K separate instruction and data cache, MMU, FPU, bus snoops;

68331—added telecom features, also DMA and other features;

68340—essentially 68332 but with simple counter/timers; and

68EC040—68040 without FPU, MMU.

Table 1 also claims that the 68000 architecture has only one implementer. I believe the Phillips/

Signetics SCC68070 "highly integrated microprocessor" is a contradicting example.

I hope that *Micro* will be more successful in the future at ensuring that its content is accurate and not overtly biased.

Joseph M. Schachner
Spring Valley, NY

Reply:

Thank you for your comments regarding the P1754 article I wrote. I agree that my table may have been a bit misleading.

The point I was trying to make was not that other architectures don't have variations but that they have very limited variations based on available devices. The implementation of P1754 is open. P1754 may be implemented as a low-performance, low-cost device or as a high-performance superscaler.

Even so, I believe the 68K family itself is large, and each architecture variation (68000, 020, 030, 040, ...) is unique. P1754 is based on the Sparc family of RISC processors. This family also has different variations: Version 7, P1754, Version 8. Picking one variation of the Sparc family that has been made a standard and comparing it to one member of the 68K family would be a more appropriate comparison.

You see, the concept of P1754 is very different from the 68K family (for that matter, from most microprocessor architectures). P1754 defines an Instruction Set Architecture (ISA) and not a device. And as such, the ISA is mostly flexible in performance grades and implementations (i.e., cache, external bus structure, basic technology).

Again, it was not my intention to insinuate that other architectures (specifically 68K, 88K, i860) are not as good or less powerful but to point out how different the definition of P1754 is.

Rudolf Usselmann

In the mailbag

(LK: liked, DLK: disliked, LTS: like to see)

Many readers continue to write about the split articles. They get the answer in this issue. From this experiment, we learned that readers do care about *Micro*. Thanks, and please continue to write and to care!—D.D.C.

December 1990

LTS: I suggest you send all the subscribers the index that appears in the issue of December of each year on a floppy disk.—J.L.B., Montevideo, Uruguay (Your suggestion sounds reasonable to me. If it proves cost-effective, I would propose it.—D.D.C.)

LK: Your articles on the ESPRIT project; what ... about the Japanese project for the fifth-generation com-

puter? LTS: Articles (cluster of articles) on other projects like those under the ESPRIT umbrella (even from Japan....)—B.M., Timisoara, Romania

February 1991

LK: "The Drive to the Year 2000"—F.P.B, Laurel, MD

DLK: The new practice of breaking up the articles into widely separated fragments is very poor.—S.L.P., Darien, CT

April 1991

LK: Analysis of multicomputer/multiprocessor systems; DLK: jumping across articles; LTS: the architecture of MIPS R4000, FPU MC68882.—K.V., Bangkok, Thailand

DLK: Your new magazine format is terrible!—K.R., North Amherst, MA

LK: The article on communication latency; it was well presented; LTS:

more articles analyzing the architecture/ performances of various machines.—A.S.M., Secunderabad, India

LK: "The end of the 386 monopoly"—the reportage style was clear and interesting; LTS: in the Micro Law column, an article on patent law changes with respect to micro applications.—D.S., Philip, Australia

LK: Micro News, Micro Review, Micro View, Micro Law, New Products. I wait for "Hot Chips." DLK: Hated: split articles! Yes, I have read the editor's reply on p. 2. Personally, I would prefer no color page at all to a split, (sliced and mixed) magazine. I'm sorry to say that.—T.P., Warsaw, Poland

June 1991

LK: Micro News... W.M.S., Atlanta, GA

NEW Conference Proceedings from IEEE Computer Society Press

12TH REAL-TIME SYSTEMS SYMPOSIUM

The proceedings covers state-of-the-art research and key developments in real-time computing and discusses issues such as hard real-time communications, priority scheduling, object-oriented modeling, optimization, real-time databases, requirements specification models, and adaptive real-time systems. Its articles discuss new research on improvements in the construction of real-time systems and to enhance the growth in real-time computing R&D.

320 PAGES. DECEMBER 1991. SOFTBOUND. ISBN 0-8186-2450-7.
CATALOG NO. 2450 \$70.00 MEMBERS \$35.00

8TH TRON PROJECT SYMPOSIUM

The TRON Project Symposium features four sessions providing detailed explanations of ITRON (industrial), BTRON (business), CTRON (central and communications), and TRON-specification VLSI CPU. It includes 19 articles covering implementation and application issues, and explores new specification proposals for next generation products based on the TRON architecture. In addition, it investigates the current efforts by researchers to develop a standardized and open architecture covering the area from operating system to VLSI chips.

264 PAGES. NOVEMBER 1991. SOFTBOUND. ISBN 0-8186-2475-2.
CATALOG NO. 2475 \$60.00 MEMBERS \$30.00

VLSI DESIGN 1992 — 5TH INTERNATIONAL CONFERENCE ON VLSI DESIGN

The proceedings of *VLSI Design '92* provides an extensive discussion of the latest advances in technology and explores recent technical opportunities in electronic design automation, and consists of over 80 papers on advanced test methods, physical design, design and fabrication, logic and high-level synthesis, VLSI tools and technology, testing, VLSI architectures, control and data path synthesis, signal processing applications, design for testability, layout, verification, industrial applications, and design verification.

400 PAGES. JANUARY 1992. SOFTBOUND. ISBN 0-8186-2465-5.
CATALOG NO. 2465 \$90.00 MEMBERS \$45.00

1ST INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED INFORMATION SYSTEMS

This book includes over 40 articles on current applications, and is divided into 12 sections that investigate information systems technologies such as disk replication and arrays, object-oriented systems, joins, parallel logic, transitive closure and synopsis, retrieval, distributed systems, architectures/parallel processing, file systems, shared memory systems, and shared nothing systems.

320 PAGES. DECEMBER 1991. SOFTBOUND. ISBN 0-8186-2295-4.
CATALOG NO. 2295 \$64.00 MEMBERS \$32.00



To order call toll-free 1-800-CS-BOOKS





Send information for inclusion in Micro News
one month before cover date to Managing
Editor, IEEE Micro, PO Box 3014, Los Alamitos,
CA 90720-1264.

Intel sues AMD for PLA copyright infringement

Richard H. Stern, Contributing Editor

The first significant copyright infringement suit based on a hardware device, other than code stored in a ROM chip (see box on copyright cases), was filed in federal court in Austin, Texas, on October 9, 1991. Intel Corp. sued Advanced Micro Devices for infringing on a copyright in what Intel identified as a "computer program stored in PLA [programmable logic array] of 80386 microprocessor" chip.

(In another part of Intel's complaint, it accused AMD of infringing on an Intel copyright in the microcode of the same chip. Intel and AMD are also engaged in patent, copyright, and mask work infringement litigation in federal court in San Jose,

California, over 80286 and 80287 chips. In addition, they have been engaged in an arbitration proceeding, with related litigation, over their chip technology interchange and second-sourcing agreements.)

Control program. Intel's complaint against AMD states that the 386 PLA contains the "control program" for the 386 DX and SX chips. Intel did not describe the function of the so-called control program (the company's designation; it is not a recognized term of art), but it appears that this PLA acts as a decoder for macro instructions. That is, an assembly-code instruction, such as "Move contents of register A to register B," may be executed by five or more steps represented in microcode. The microcode ROM of a microprocessor such as the 386 stores the instructions for these steps, as well as the steps for

PLA and ROM copyright cases

In 1987 Alloy Computer Products brought two suits in federal court in Los Angeles, charging copyright infringement of an alleged computer program stored in a PLA device.^{1,2} Apparently, these cases never came to trial, perhaps because the defendants surrendered without contesting the claim. No reported precedent on copyright in PLAs has emerged from these suits.

Federal case law has established that operating system software stored in ROM is subject to copyright protection.³ This principle apparently extends to microcode sorted in ROMs in microprocessors. However, the one decision on that issue concluded that the particular microcode involved (8086/8088) was

not infringed because the accused microcode (V20/V30) was too different to be found "substantially similar" (the test for copyright infringement).⁴

References

1. Alloy Computer Prods. v. Ultratek Corp., No. 87-6993 (C.D. Cal. filed Oct. 20, 1987).
2. Alloy Computer Prods. v. Asadi, No. 87-1285 (C.D. Cal. filed Mar. 23, 1987).
3. Apple Computer, Inc. v. Franklin Computer Corp., 714 F. 2d 1240, (3d Cir., 1983); cert. dismissed by stip., 464 U.S. 1033 (1984).
4. NEC Corp., v. Intel Corp., 10 USPQ 2d 1177 (N.D. Cal. 1989).

other instructions.

What Intel describes in its complaint as the control program may therefore be a symbolic notation for a decoder PLA that receives as input the object code for an expression such as, "Move contents of register A to register B." The decoder PLA then provides as output a starting address in the microcode ROM for the relevant five or more steps. Such a decoder would be, essentially, an array of gates (such as And and Or gates) corresponding to a set of Boolean equations for transforming instructions of the sort described (placed into object code format) into appropriate addresses of the microcode ROM.

PLA contents are usually specified symbolically. This symbolic representation is presented to the device that creates the hardware representation. Since the symbolic representation describes the relationship between inputs and outputs of the PLA, essentially Intel is asserting a copyright in what the PLA does, rather than the specific way it does it. Query: Can this be meaningfully distinguished from a copyright in a source code, which is copyrightable?

There are no close legal precedents on whether information in this form is a "computer program," as that term is defined in the Copyright Act. (Section 101 of the Copyright Act defines a computer program as a set of statements or instructions used to bring about a result in a computer.)

There are also no close legal precedents on whether making and selling a physical device—for example, a 386 clone—containing a substantially similar array of gates infringes on a copyright in the information describing the array. Such copyrights are registered with the Copyright Office by filing the information in some sort of printout (perhaps the Boolean equations, or an object code printout in binary or Hex of the microcode addresses tabulated against the object code inputs). Query: Is the physical device a legally protected "copy" of what was registered?

Semiconductor Chip Protection Act

The US Copyright Office's refusal to issue copyright registrations for semiconductor chip layouts led to passage of the Semiconductor Chip Protection Act of 1984. This law covers layouts of unprogrammed PROMs—programmable ROMs—but apparently does not cover layouts, or mask works, of programmed PROMs blown from such unprogrammed PROMs. It also covers layouts of both personalized and unpersonalized gate arrays. Probably, it would cover the layout of the PLA involved in the *Intel v. AMD* case. But a functionally similar PLA with a different layout from the registered layout would not be covered by such a registration.

Did the Copyright Office know?

Intel attached the copyright registration certificate (No. TX 3 121 803) for its PLA to its complaint. Such certificates include a "correspondence" box that the Copyright Office checkmarks if correspondence occurred between the office and the registrant. The box on this certificate is not checked.

That omission may suggest that the Copyright Office rubber-stamped the application without realizing that a PLA was involved (assuming that the Copyright Office even knows what a PLA is), and without realizing that the computer program in question was perhaps imaginatively so designated. One might expect the Copyright Office, if it knew that a gate array was being claimed as an information storage device, would ask for an explanation.

Further, considering the Copyright Office's history of resistance to copyrights on hardware, (see box on Semiconductor Chip Protection Act) one might expect it to issue gratuitous

advice to Intel that the copyright did not extend to physical devices built in accordance with or embodying the information. It is common practice for the Copyright Office to do so, and to place a record of the advice, in the form of a letter, in the official file.

None of the foregoing suggests that an explanation about the PLA, if Intel had given one to the Copyright Office in response to such a demand, would necessarily have been unsatisfactory. There is a great deal of isomorphism between 1s and 0s in a ROM and the equivalent in a PLA. One device Ands sets of Or gates, while the other device Ors sets of And gates. It may well be that analogous assembly codes can be created and described for each device.

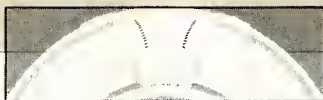
Both such codes (if they are accepted as being codes) might be compiled or assembled into 1s and 0s (connections and open circuits) in analogous ways. But one would have expected the Copyright Office to ask for such an explanation if it understood what was going on in the case of this copyright application. Perhaps AMD's eventual response to Intel's claims will shed more light on these issues.

If PLA programs are held protectable by copyright, that will have startling and important implications for designers of electronic circuitry. (For a further discussion of the issue, see Micro Law, "Field-programmable logic devices—Are they hardware or software? Can their programmed configurations be protected against copying?" in *IEEE Micro*, Vol. 6, No. 5, Oct. 1986, pp. 61-62 and 78.)

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 198 Medium 199 High 200



Guest Editors' Introduction

Database Machines: Trends and Opportunities

M. Abdelguerfi

University of New
Orleans

A.K. Sood

George Mason
University

As could be expected, demands of the user community and developments in computer hardware and software have affected database technology. Although, traditionally, database practitioners worked with well-structured or semistructured databases, the development of new and reliable sensing techniques added impetus to research of unstructured databases such as those containing images.

These trends focused interest on unconventional database applications, despite their very large data volumes. For example, a 150-bed hospital generates 2 Gbytes of image data each day, and the US National Aeronautics and Space Administration's planned Earth Observing System program could generate several Gbytes of data every day. The archiving, retrieval, and management of such large databases constitute a complex and difficult task. Recent trends in database technology suggest that database computers can best undertake the efficient management of large databases. That is, specialized hardware and software configurations aimed primarily at handling large databases and answering complex queries become the best solution.

In general, most of the data modeling effort focuses on three widely used, structured database models: relational, hierarchy, and network. Other, recently proposed object-oriented data models may be suitable for modeling databases that include structured and unstructured data. Since the theoretical underpinnings of the relational model have been well defined, the relational model became the focus of most of the commercial and research efforts in database machines. Some additional effort in the development of hardware solutions for semistructured databases like text retrieval also surfaced. The articles in this issue reflect these database machine trends. Four of them discuss issues related to relational databases,

and another delves into text retrieval.

The primary motivation of the database machine approach is to increase the performance of updating operations and the query processing of databases. One approach taken to achieve this objective employs parallel architectures. The designer must choose the appropriate approach for data distribution and storage, interprocessor communications, and computational capability of each node. We can divide the parallel architectures studied in the context of database processing into three groups: shared memory, shared disk, and "shared nothing."

The current trend in the design of database computers is toward the increasingly popular shared-nothing¹ message-passing architectures. Designers usually prefer these architectures over shared-memory and shared-disk architectures because they permit a high degree of scalability. The MDBS database computer described by Hsiao in this issue belongs to this category.

Database engines^{2,3} represent another example of shared-nothing database systems. The high-performance and fault-tolerant database engines generally support a server/client architecture. A number of new database engine products have already hit the market, and more are currently under development. Teradata's DBC/1012 uses 80386 processors (up to 1,024). Because processors can be added as needed to increase both CPU and input/output capabilities, this database engine can handle very large databases.⁴ White Cross offers a database engine based on the Inmac transputer. The hardware and software structure in this database engine permits a maximum of 100,000 MIPS of processing power.⁵

The articles in this special issue on database computers fit into two main classes. The first class centers on hardware accelerators for database computers, and the second class presents two different database computers. The proposed hard-

Companion issue to December 1991 *Computer on Heterogeneous Distributed Database Systems*

ware accelerators concentrate on the efficient implementation of specific database tasks. The articles on database computers deal with the overall architecture, performance, and data distribution issues; they also discuss specific hardware accelerators incorporated in the systems.


The first three articles examine the design and evaluation of special-purpose accelerators for database computers. Lee et al. present two VLSI accelerators for formatted and unformatted databases. The database tasks performed by the accelerators include operations such as associative search, aggregation, and string search.

Faudemay and Mhiri present an associative accelerator whose speedup ratio is independent of the database size. The accelerator implements relational database operations and sorting and aggregate functions.

Our article describes the design and simulation of a bit-serial accelerator for statistical aggregation operations. A number of bit-serial processing elements connected according to the odd-even network topology make up the accelerator. The proposed unit achieves a high degree of pipelining and parallelism.

The remaining two articles describe two database computers and emphasize both architectural and performance considerations. Hsiao presents an experimental database computer with a variable number of database processors known as the Multiback-end Database Supercomputer (MDBS). Each micro-processor-based processor has a private database store, consisting of a small Winchester drive for paging and metadata and a large drive for the database.

Finally, Inoue et al. describe a relational database processor with specialized hardware for searching and sorting known as Rinda. Rinda is composed of two hardware accelerators, the CSP (Content Search Processor) and ROP (Relational Operation Accelerating Processor). The CSP searches rows of data on a disk storage and transfers the selected rows to the main memory. The ROP subsequently sorts the row transfers to the main memory.

We thank the reviewers who helped us referee the submitted papers and the *IEEE Micro* editorial staff. This special issue would not have appeared without their assistance. 

References

1. D.J. DeWitt et al., "A Single User Evaluation of the GAMMA Database Machine," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, Boston, 1988; also in *Proc. Fifth Int'l Workshop Database Machines*, Oct. 1987, pp. 370-386.
2. V.A. Topkar, O. Frieder, and A.K. Sood, "Duplicate Removal on Hypercube Engines: An Experimental Analysis," *Parallel Computing*, North Holland (to be published).

3. O. Frieder, V.A. Topkar, R. Karne, and A.K. Sood, "Experimentation with Hypercube Database Engines," *IEEE Micro* (to be published in Feb. 1992).
4. Teradata Corp., *DBC/1012: Data Base Computer System—Introduction*, Boulder, Colo., 1986.
5. M. Butler and R. Bloor, "Client-Server Engines," *DBMS*, June 1991, pp. 17-18.



M. Abdelguerfi, an associate professor of computer science at the University of New Orleans, actively participates in research on database and knowledge-base machines, information retrieval, database management, design and analysis of parallel architectures, and VLSI architectures for nonnumeric computations.

Abdelguerfi received the Diploma in electrical engineering from the National Polytechnic School of Algiers, Algeria, and an MS and PhD in computer engineering from Wayne State University, Detroit. He is a member of the IEEE, the IEEE Computer and Circuits and Systems societies, the Association of Computing Machinery, Eta Kappa Nu, and Tau Beta Pi.



Arun K. Sood is a professor of computer science at George Mason University. His research interests include image analysis, signal processing, parallel and distributed processing, database machines, performance modeling, and optimization. His research efforts have resulted in more than 70 publications and a patent. He guest edited the *IEEE Transactions of Systems, Man and Cybernetics* special issue on unmanned systems and vehicles.

Sood received a Bachelor's degree from the Indian Institute of Technology in Delhi and MS and PhD degrees from Carnegie Mellon University, all in electrical engineering. He is a member of the IEEE Systems, Man, and Cybernetics Society's Administrative Committee.

Address questions concerning this issue to M. Abdelguerfi, Department of Computer Science, University of New Orleans, New Orleans, LA 70148; mahdi@noac.cs.uno.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

High 150

Medium 151

Low 152

VLSI Accelerators for Large Database Systems

VLSI accelerators speed up time-consuming database operations while maintaining the cost and flexibility benefits of general-purpose computers. An experimental VLSI relational data filter performs high-speed associative searches, and a text filter searches for strings at up to 1 Gbyte/s.

Kuo Chu Lee

Takako Matoba Hickey

Victor W. Mak

Gary E. Herman

Bellcore

Attempts to improve the performance of query processing generally fall into one of two categories. Application-specific, or dedicated, database machines¹⁻⁶ improve a specific set of applications and so usually do not support other types of database applications or general-purpose computing. These machines are expensive to develop and maintain, so they are only economically feasible where the cost can be justified.

General-purpose database systems⁷⁻⁹ avoid the problem of specialized hardware by developing database system software on general-purpose computers. But they are less efficient than dedicated machines with customized I/O and processing units.^{4,10,11}

A VLSI accelerator approach can improve the performance of database processing on general-purpose computers. We identified the most time-consuming database operations on general-purpose computers and designed application-specific accelerators to speed up these operations. By using a rapid-turnaround design methodology,¹² we can correctly and efficiently design VLSI accelerators.

Associative search, aggregation, and string search are among the lengthy operations we identified. Associative searches and aggregations are performed on formatted relational records consisting of multiple attributes of specific types and lengths. The associative search selects records

based on a user-defined predicate that specifies the relationships between multiple pairs of attributes and search patterns. The aggregation operation sums or counts an arbitrary attribute of all the selected records. The string search finds all occurrences of a pattern in an unformatted data string.

All of these operations require scanning a large portion of data from the storage devices. As a result, the evolution of mass storage technologies such as optical disks¹³⁻¹⁵ and silicon memory storage¹⁶ has strongly influenced the fundamental design assumptions for VLSI accelerators. These advanced device technologies make possible data transfer rates of up to a few Gbytes/s. At such high I/O rates, direct transfer of data from storage to the CPU is likely to cause thrashing. The CPU cannot perform useful work while it handles frequent page faults caused by the increased I/O operations. Therefore, a critical function of the VLSI search accelerators is to exclude as much irrelevant data as possible before delivering data to the CPU. CPU cycles can be saved for other general-purpose tasks, such as index traversal, updates, transaction processing, and statistical analysis.

We designed and had fabricated an experimental VLSI accelerator research prototype for relational data filtering. This relational data filter performs high-speed associative search and aggregation operations for formatted databases. For unformatted databases, we designed an

experimental fast string search VLSI accelerator that provides a few orders-of-magnitude improvement in text search speed. The VLSI accelerators have precise instruction sets and hardware interfaces that facilitate their integration into general-purpose computer systems and dedicated systems.¹⁷

Formatted database accelerator

The experimental relational data filter speeds up formatted database operations by supporting associative search and aggregation operations. Speeding up the associative search is not a simple task since it requires either a full database scan or a full indexing scheme. A full scan of the database in a general-purpose computer architecture is very costly. Full indexing requires large memory space—at least proportional to the number of data items stored—and incurs large overhead for update and database administration. The full indexing scheme works well for selection queries based on indexed attributes. It is ineffective, however, for complex queries that require scanning a large portion of the database, such as range queries, selection queries with *don't care* characters in the criteria, and queries that take statistical measures on some attributes. Thus, improving the performance of associative searches is a fundamental need in formatted database systems.

We approached the problem of speeding up the full database search operations by using a VLSI relational data filter that scans directly at the high-speed storage output. In support of associative search operations, the data filter selects interesting records from the storage output and passes only those to the main memory. This allows the general-purpose processor to concentrate on more complex operations on the smaller set of records. The data filter also provides high-speed database aggregation operations such as COUNT, SUM, and MAXIMUM, based on arbitrary selection criteria.

Architecture. The VLSI data filter can be viewed as a reduced instruction-set computing (RISC) processor,^{18,19} with the instruction set optimized for associative search and aggregation operations. A more complex comparator, such as an ALU, is needed to enable the filter to perform more complex aggregation operations. However, multiple complex comparators are too costly. The architecture as illustrated in Figure 1 achieves parallelism by efficiently sharing the single comparator unit across multiple queries stored in the instruction buffer.

The data filter accepts records continuously from the storage output, synchro-

nizes the stream speed to the internal speed, and holds each record in an internal record buffer. Records are double-buffered: One record is resident in one buffer and is examined, while the next record is arriving in the other buffer. Instructions are also double-buffered: Instructions in one buffer are executed against a series of records, while the next batch of instructions is loaded into the other buffer. Selected portions of the resident record are evaluated in the execution unit (an ALU and its associated logic) according to the instructions in the instruction buffer. If any of the instruction sequences are evaluated to be true, that is, if the record satisfies a selection query, the ID of the query passes off the chip to the main memory. If any queries are satisfied, the record passes to the main memory at the end of the execution time interval.

The on-chip instruction and record buffers improve the speed of instruction and operand fetch and thus improve the performance of the associative search. Since an associative search requires repetitive application of the same query against the search attributes of all the records in a relation, a reduction in instruction fetch time significantly reduces the overall search time of a single query. Reducing the operand fetch time allows more instructions to be applied against each record; thus, the apparent search parallelism of the data filter is also improved.

Pipelining further improves the apparent execution rate and search throughput by overlapping operations in different stages of the data filter. For example, our prototype

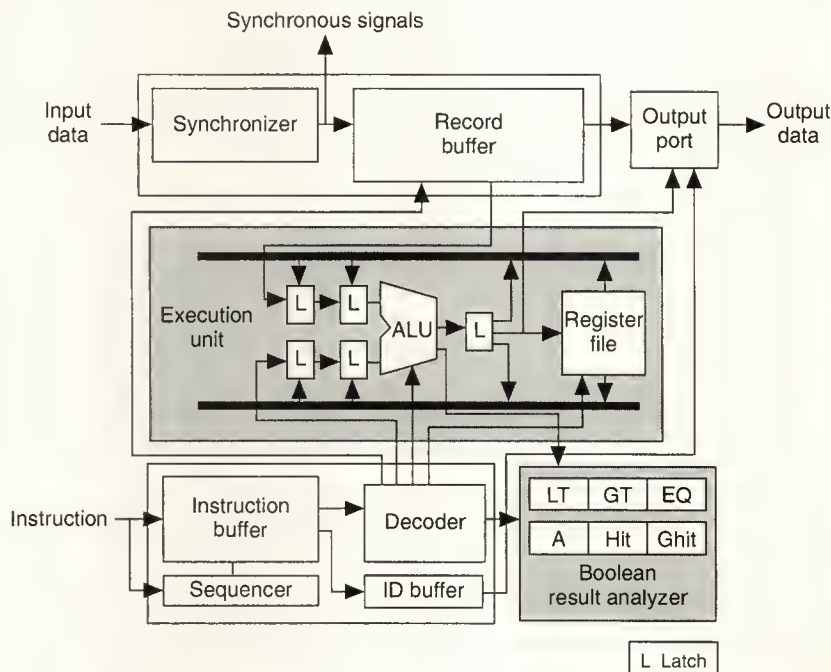


Figure 1. Experimental data filter architecture.

executes each instruction in six pipeline stages including instruction fetch, instruction decode, operand fetch, ALU execution, register store in parallel with Boolean evaluation, and output of query ID.

The size of the record buffer, instruction buffer, and data path can be engineered to suit the application need and technology used. We targeted our prototype for 64 Mbytes/s for the record input stream and 62.5 ns for the processor cycle time. With a 32-bit data path and a 128-byte record buffer, we determined that the instruction buffer holds 32 instructions, the maximum number of instructions that can be executed in the record interarrival time of 2 μ s. A longer record size can be supported by using external buffers. The portion of data specified in the search predicates are stored in the record buffer. We expect that the search attributes for a typical relational record will not exceed 128 bytes.

There are four major blocks inside the data filter.

- **Record buffer control.** The record buffer control synchronizes the arrival of incoming records, manages the double buffering of records, and synchronizes signals for the external control processor. The synchronizer detects only a few special words such as *start of record*, *end of record*, *start of segment*, and *end of segment*. When it finds any of these words, the synchronizer outputs an appropriate boundary indicator and adjusts the address register to place the incoming information in the appropriate location within the record buffer. With minimal embedding of schema information in the data stream, the data filter hardware is free from the runtime decoding of schema information that can slow down filter processing.
- **Instruction buffer control.** The instruction buffer control manages the double buffering of instruction batches, sequencing, and decoding. We kept the logic of instruction buffer control simple by writing a simple instruction set, as discussed later. Without branch instructions, the sequencer becomes a simple counter with a reset logic. The small and orthogonal instruction set enables simple and fast decoder implementation using programmable logic arrays (PLAs).
- **Execution unit.** A parallel data path with an ALU and a register file make up the execution unit. Two operands are fetched into two operand receiving latches and then transferred to the ALU operand latches. By using these two close-coupled latches, the operands receive sufficient set-up time for the latches at the ALU. As a result, the maximum stage delay of the pipe is very close to the clock cycle time if a two-phase, symmetric clock is used. The result of the ALU operation moves from the destination latch to the register file. The operand can be selected from the register file, from the record buffer, or from the pattern specified by the instruction. All the com-

SELECT record FROM relation WHERE boolean expression
FOR ID

where boolean expression ::= term | term OPb boolean
expression

OPb	::= \wedge \vee
term	::= attribute OPa value
OPa	::= $>$ $<$ $=$ \geq \leq \neq

Figure 2. Boolean select operation.

parison and data movement operations are carried through the ALU. In the case of a data dependency, the temporary register can be used to store the result of the first ALU operation and to direct it to the ALU input latch for the next instruction to save translation time and register locking overhead.

- **Boolean result analyzer.** The most important feature of the experimental data filter is the fast Boolean predicate evaluation and the conditional assignment capability defined in the instruction set and supported by the internal architecture. Two single bits, A and Hit, implement the continuous evaluation of conjunctive normal form and conditional assignment that we use to avoid conditional branch instructions, as we discuss later. A comparison result derived from the ALU flags and the test condition is first evaluated with a Boolean accumulator A under the opcode control. Then, the result of the evaluation is stored back into the A accumulator. If the result of the last Boolean operation pertaining to a query is true, a Hit flag is set. This Hit flag is used by the conditional assignment instructions. If it is set, the result of the assignment will be written into the register file; otherwise, the result is discarded.

Instruction set. The instruction set of the data filter blends complex instruction-set computing (CISC) and RISC philosophies. Each instruction exhibits high semantic content and performs multiple actions, as advocated in CISC. However, the number of different instructions is minimized to simplify the instruction decoding, as advocated in RISC. The instruction set has two major modes, one to improve the associative search operations and the other to enable aggregation operations.

Boolean mode. The primary objective of the data filter is to speed up the record selection operations. The data filter takes fixed-size relational records²⁰ as input data and relational queries as instructions to be executed against the input records. Records that satisfy the query predicates become outputs. The basic format of the select operation is given in Figure 2. As an example, the query, "What Indian restaurants are in New York?", may be written in relational algebra.


```

SELECT record FROM (relation = "restaurants")
WHERE (type = "Indian") ^ (location = "New York")
FOR (Query 1)

```

The design of the instruction set to realize this selection predicate directly affects search performance. For example, if we assume each string can be coded into one comparison unit, say a long integer, we might implement this query in a general-purpose processor as

```

addr0:  cmp    @relation, #restaurants
        jneq   addr1
        cmp    @type, #Indian
        jneq   addr1
        cmp    @location, #New York
        jneq   addr1
        mov    hit, #1
addr1:  ...

```

Three comparisons correspond to each of the comparisons in the select clause with each followed by a branch instruction.

To improve execution efficiency, the data filter architecture includes Boolean instructions. These instructions speed up the selection operation through the use of the Boolean accumulator A, which accumulates the Boolean result pertaining to one query. Three basic operations can be performed on the Boolean accumulator: PUSHA, which pushes a Boolean value to A; ANDA, which Ands another Boolean value to the value in A; and ORA, which Ors another value to the value in A.

The prototype only supports a one-bit accumulator, but it can be easily extended into a stack to support a wider range of truth operations, such as $(B \wedge C) \vee (D \wedge E)$, where B , C , D , and E are Boolean truth values.

In addition to the three basic operations, four operations on A are defined to tie together the truth values pertaining to one query. FANDA and FORA are like ANDA and ORA but are used for final instructions pertaining to one query. These two instructions turn on the Hit flag, which can be used as a condition for arithmetic instructions discussed in the next section. SANDA and SORA are also used for final instructions, but these turn on the Ghit flag, which causes the output of the current record.

The basic format of a Boolean mode instruction is given by

OPr(OPa(MEM, Pattern, Mask), ID).

where OPr is the operation on accumulator A just discussed. This one instruction specifies three operations to be executed in sequence: a comparison, a test on comparison result, and an operation between the result and the accumulator A. The data filter first compares the word at address MEM (mask)

with Pattern, and, if the comparison flags (LT, GT, EQ) satisfy the flag pattern specified by OPa, the filter sets the Boolean value to true. Finally, it performs the OPr operation on this Boolean value.

A select query may be decomposed into multiple data filter instructions, one instruction for each of the attribute comparisons. For example, the selection query previously presented may be decomposed as

```

PUSHA(EQ(relation, "restaurants", f), 1).
ANDA(EQ(type, "Indian", f), 1).
SANDA(EQ(location, "New York", f), 1).

```

Thus the data filter executes the selection query in only three machine cycles. We intentionally avoid using the exact format here to better illustrate our idea. A similar format can be used as an input to a front-end compiler that compiles the input into the binary form understood by the actual filter. In the example, we use strings such as "relation" to represent memory addresses and use mask f to mean we want the entire attribute.

Arithmetic mode. The second objective of the experimental data filter prototype is to support aggregation operations such as COUNT, SUM, and MAXIMUM. Adding this capability to the filter offloads the main CPU from loading and scanning large amounts of data. It also allows simultaneous execution of aggregation operations that cannot be easily achieved in conventional database systems.

Aggregate operations can be supported in the data filter by adding a conditional assignment capability that allows accumulating the result to a register depending on certain conditions. A conditional assignment is conventionally implemented by a comparison followed by a branch followed by an assignment instruction. However, this approach can be inefficient due to the pipeline breaks. Hence the data filter defines an instruction mode that can perform the ASSIGNMENT operation based on a condition code. This type of instruction together with the Boolean instructions previously discussed allow numeric operations to be carried out without pipeline breaks.

The basic format of the arithmetic mode instruction is given by

OPalu(Cond, M1, M2, M3).

The semantics of the instruction are: If the condition specified by Cond is true, carry out the ALU operation OPALU on two operands M1 and M2 and store the result into M3. The condition can be true (unconditional), hit (set by Boolean instructions), and six combinations of three ALU flags: less-than, greater-than, and equal. M1 can be a memory address or a register, M2 can be an immediate operand or a register, and M3 is a register.

The main loop of the counting query, "How many people live in San Francisco?", can be compiled as

```
PUSHA(EQ(relation, "person", f), 2).
FANDA(EQ(town, "San Francisco", f), 2).
INCB(Hit, null, sumreg, sumreg).
```

A sum query can be formed by replacing the increment operation with an add operation on the summing attribute, say salary, with the sum register.

```
ADD(Hit, salary, sumreg, sumreg).
```

A maximum query can be formed by replacing the increment operation with a compare operation on a maximizing attribute, say age, with the maximum register followed by a conditional assignment of the maximizing attribute to the maximum register.

```
COMP(Hit, age, maxreg, null).
EQUA(Gt, age, null, maxreg).
```

We can extend operations on accumulator A so that the accumulator can be set or cleared based on application-specific condition flags for linking computation results of added functional components. For example, we can add a string search unit, which we describe later, to the data filter as a functional component. The result of a string search operation, finding a pattern in the input stream, can be kept in a condition code and used to set or clear the accumulator.

Discussion. The programmability of the filter lends it to general applicability in a variety of applications requiring high-speed filtering of structured data. Further, the filter architecture scales well with improved fabrication technology, supporting a faster evaluation rate and more complex query sets for finer fabrication geometries.

The VLSI data filter performs better for synchronous search of structured data than both CISC and RISC general-purpose microprocessors. This advantage derives primarily from the architectural support provided both for pipelined Boolean predicate evaluation and for efficient I/O through double buffering of data. We avoided pipeline breaks by using Boolean evaluation and conditional assignment instructions. The parallel data path for instruction and immediate operand store prevents a pipeline stall due to double-operand fetch contention. These features permit more pipeline stages while avoiding pipeline breaks. The architectural optimizations are combined with the overlapped cache refill operations provided by the multiported record buffer and instruction buffers. Together they allow the VLSI data filter to perform synchronous data search faster than a conventional microprocessor used in the same technology.

We made the data filter more efficient by integrating sev-

eral subsystems on a single VLSI chip. The experimental VLSI data filter includes additional subsystems beyond the microprocessor itself, including a tri-port RAM for the record buffer, dual-port RAM for the instruction buffer, and extensive glue logic to support synchronization, instruction sequencing, and handshaking. Implemented off chip, these functions would require expensive high-speed components to perform as well as the integrated design. We estimate that it would take four times as much hardware to replicate the data filter and its support functionality compared to our prototype design. The improvements in execution efficiency, combined with the factor-of-four reduction in hardware, provide a bias for advocating the use of a VLSI accelerator over an approach based on a general-purpose microprocessor.

Designers can achieve the performance advantage with only a modest effort by taking advantage of high-level specification languages and VLSI tools. Matoba et al.¹² describe a rapid-turnaround methodology that we used to realize our data filter prototype in nine staff-months. The ability to design quickly a VLSI system as complex as the data filter should allow custom VLSI to be commercially viable in many more application systems than it has been. The ability to explore options rapidly with precision should allow systems research to progress rapidly beyond the paper design stage where ideas often stall for lack of expertise and/or human resources.

The VLSI research prototype fabricated in 2- μ m CMOS technology executes more than 16 million predicate evaluation instructions per second and achieves 64-Mbyte/s search throughput for individual queries or sets of queries comprising a total of up to 30 Boolean predicates. Table 1 summarizes some chip parameters for our first prototype. The chip consists of 91,000 transistors and the area is 475 mils square. The chip's speed is determined by the longest stage in the pipeline (54 ns, determined by the speed of the ALU cell). The first wafer lot produced 33 working chips, a yield of 16 percent. The prototype filter is an integral part of a working experimental prototype database system that efficiently supports a rich set of query types plus full transaction semantics.¹⁷

Table 1. Experimental chip parameters.

Number of transistors	91,000
Chip size	476 \times 477mil ²
Number of pins	171
Cycle time	54 ns
Power at 18.5 MHz	2.5 W
Supply voltage	5 V
Process technology	2 μ m CMOS

Unformatted string search accelerator

String searching is another operation that requires extensive scanning of unformatted data in database applications. The problem of string searching is to find all occurrences of a p -character pattern P constructed from a vocabulary of m distinct characters in an s -character data string S . The pattern P may also contain *don't care* characters. For typical applications, $p \ll s$ and $m \ll s$. Since the size of the data string is usually very large, sequential search via general-purpose processors is prohibitively slow. Thus, reducing the time required to search a large text database has been an active area of research for the past decade.

We propose a new parallel VLSI algorithm called Data Parallel Pattern Matching (DPPM) and a corresponding VLSI document-search engine called DPPME. The DPPM algorithm differs from most previous work in that it serially broadcasts each character in the pattern and compares the pattern to a block of data in parallel. The DPPM algorithm uses the high degree of integration of VLSI technology to process quickly through parallelism. Based on simulation statistics and timing analyses of the hardware design, we can achieve a search rate of multiple Gbytes/s DPPME using advanced VLSI technology.

The DPPM algorithm. Let $S[1:n]$ be the data string of n characters to be searched and $Pat[1:p]$ be the pattern of p characters. The data string is divided into blocks of b characters each and searched a block at a time. Let $Blk[1:b]$ be the current data block of size b characters. Basically, the DPPM

algorithm serially broadcasts each pattern character to a block of the data string. If the pattern character matches with any of the characters in the block, the next pattern character is broadcast in the next comparison cycle. If, at any cycle, no match is found between the current pattern character and the data block, and if no partial match is carried over from the previous block, the filter discards the data block. The search continues with the next block. A partial match occurs when $Pat[i]$, $i < p$, matches with the last data block character $Blk[b]$. This partial match information is stored and used in the next block to continue the search by comparing $Pat[i+1]$ to the first data block character $Blk[1]$.

We can best illustrate the DPPM algorithm with a simple example. Suppose we conduct a search for the pattern "abcd" in the data string "abadbbabcedee." Figure 3 shows the operation of the DPPM algorithm using a block size of four.

The first block, containing the characters "abad," is first loaded into the comparator array. When compared to the first pattern character "a," two matches are detected. The second pattern character "b" is then broadcast and compared to the characters to the right of the matched characters in the previous cycle, that is, "b" is compared to the second and the fourth characters in the block. Since a match is detected again at the second character, a comparison of the third pattern character "c" with the third character in the block is necessary. This time, no match in the block is observed; thus, the current block is discarded and the search continues with the next block. This early mismatch detection mechanism avoids broadcasting and comparing the fourth pattern character "d" to the current block since this comparison is unnecessary.

The next block contains the characters "bbab." The pattern compares successfully up to the second character "b." At this point, the end of the block is reached. The pattern may span the block boundary. DPPM acknowledges this partial match information and continues the match in the next cycle. Since no other match occurs in this block, the current block is also discarded.

The third block contains the characters "cedee." The first pattern character "a" has no match with the block. At this point, DPPM acknowledges a partial match in the previous block up to the second pattern character. Therefore, it jumps to the third pattern character "c" and continues the partial match from the previous block. Finally, a hit or an occurrence of the pattern is detected with the fourth pattern character "d."

Although not shown in this example, the DPPM algorithm can also detect multiple occurrences of the pattern even if they overlap, and no backtracking is required to detect all occurrences.

Figure 4 on page 14 shows the pseudocode of the control flow of the DPPM algorithm. $DC[1:p]$ is a bit-vector indicating the *don't care* positions in the pattern. $DC[i]$ is set if there is a *don't care* at $Pat[i]$. $Mask[1:b]$ controls the activation of the

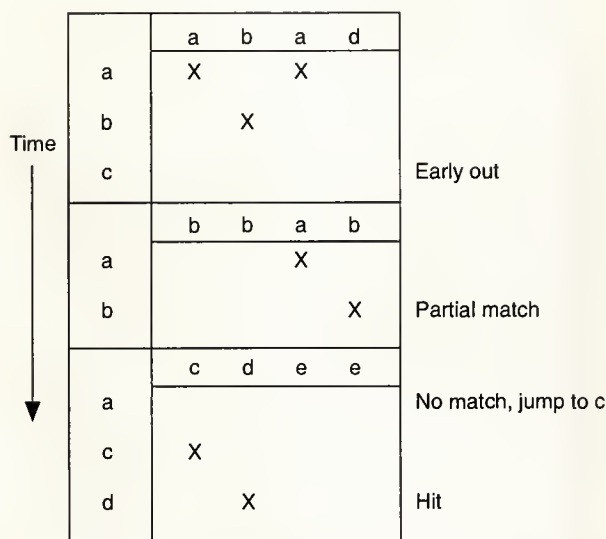


Figure 3. DPPM example.

comparator array based on the results of the previous comparison cycle. If $Pat[i]$ matches $Blk[i]$ in the first comparison cycle, then $Mask[i+1]$ is set in the next comparison cycle enabling the comparison between $Pat[2]$ and $Blk[i+1]$. $T[1:b]$ holds the results of the comparator array. $Vin[2:p]$ and $Vout[1:p-1]$ hold the partial match information from the previous block and the current block, respectively. $Vin[i]$ is set if a partial match is found in the previous block up to and including the character $Pat[i-1]$. $Vout[i]$ is set if a partial match is found up to and including the character $Pat[i]$ in the current block.

With the use of Vin and $Vout$, partial match can be continued in the next block without any backtracking of the data string. Since the algorithm is independent of the block size, the search rate can be increased simply by increasing the block size, or the number of comparators. As a result, the DPPM algorithm can efficiently use the high degree of integration of

```

while (← End of Data String) do begin
  GetNextBlock(Blk);
  forall i from 1 to b do Mask[i] := TRUE;
  forall i from 1 to (p-1) do Vout[i] := FALSE;
  i := 1;
  while i ≤ p do begin /* Comparison Cycle */
    forall j from 1 to b do
      T[j] := Mask[j] ∧ (DC[i] ∨ (Blk[j] = Pat[i]));
    if (i < p) then Vout[i] := T[b]
    else begin /* i = p: Report hit found */
      forall j from 1 to b do
        if (T[j]) then Report Hit at j;
      break; /* Match(es) in block */
    end;
    if (V[1] ∨ T[1]) then begin
      /* Prepare Mask for next comparison cycle */
      forall j from 2 to b do
        Mask[j] := T[j-1];
      Mask[1] := Vin[i+1];
      i := i + 1;
    end
    else if (V[i+1] ∨ Vin[i]) then begin /* Continue partial match */
      /* Skip Unnecessary Pattern Characters */
      do i := i + 1 until (Vin[i]);
      Mask[1] := TRUE;
      forall j from 2 to b do Mask[j] := FALSE;
    end
    else /* Early Out */
      break;
  end;
  forall i from 2 to p do Vin[i] := Vout[i-1];
end;

```

Figure 4. Pseudocode of the DPPM algorithm.

VLSI technology to attain high-speed processing through parallelism.

VLSI prototype design. Figure 5 shows the circuit block diagram of the experimental DPPM engine. Before the actual search operation, the pattern, pattern length, and *don't care* (DC) positions are first loaded into their corresponding registers. The data string is buffered and read one block at a time to the Block register. The comparator array performs the actual comparison between the pattern character and the data block. The results of the comparator array are Anded with the mask to form register T. The DPPM engine integrates the mismatch detection and the partial match propagation mechanisms by combining the Vin register (partial match information from previous block) with the old T register (match results of the previous comparison cycle) to form the mask for each comparison cycle. The first bit of the mask is from $Vin[i]$, and the last (b-1) bits are from the first (b-1) bits of T in the previous cycle. The last bit of T is stored into $Vout[i]$. Each time a new data block is read, the first (p-1) bits of $Vout$ are loaded into the last (p-1) bits of Vin . The first bit of Vin is always set.

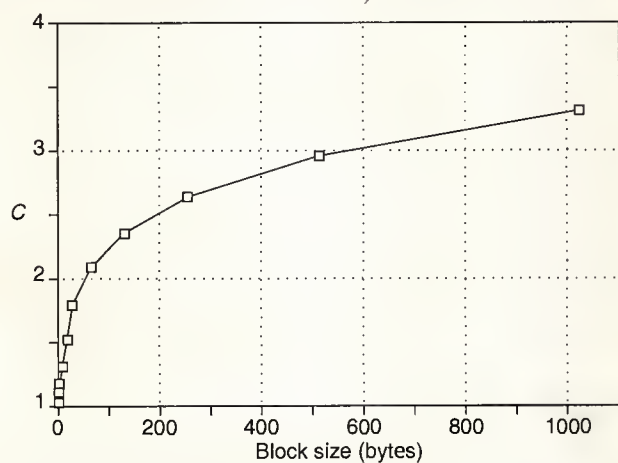
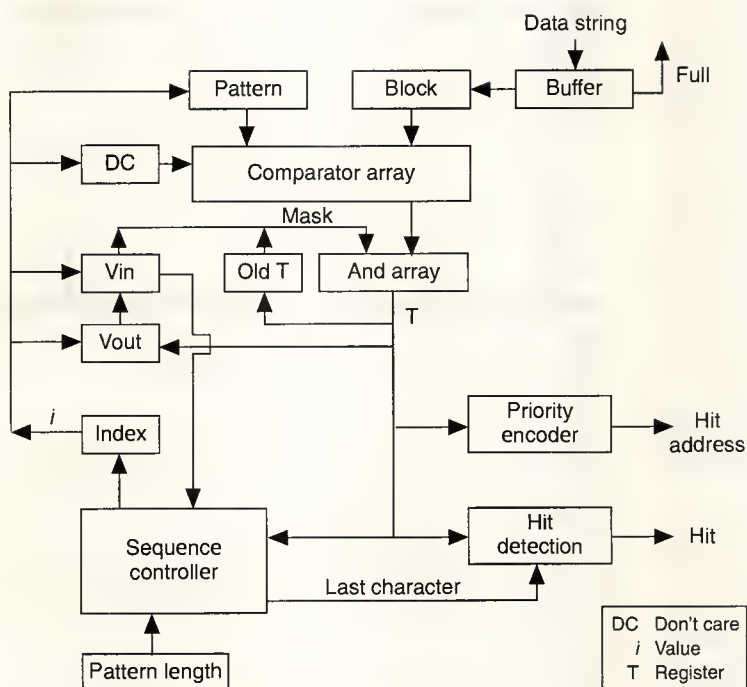
The sequence controller controls the operation of the DPPM engine by generating the value i , which is used to index the pattern, *don't care*, Vin , and $Vout$ registers. By monitoring the values of T, the content of the Vin register, and the pattern length, the sequence controller decides for each cycle one of the following three actions:

- Compare the next pattern character with the current block,
- Jump to a pattern character to continue the partial match from the previous block, or
- Discard the current block and continue the search with the next block.

Step 1 is taken if this is not the last pattern character and the content of T, that is, the number of matches detected in the current cycle, is nonzero. If T is zero, then the index of the next pattern character to be used for comparison is determined by a priority encoder that encodes the first nonzero bit in the Vin register after masking off the first (i-1) bits of the Vin register using a linear shift register. Step 3 is taken if the last pattern character is reached, or if T is zero and there is no more partial match propagation from the previous block (early out).

The sequence controller also generates a last character signal when the last pattern character is reached. This signal is used by the hit detection unit, which checks the values of T to report any hit in the search. The priority encoder produces the encoded addresses for all hit positions in the current block.

The critical path of the circuit is from the pattern register, through the comparator and And arrays, to the priority encoder. Using 2- μ m CMOS technology, we achieve the comparison cycle time of about 50 ns for a block size of 1,000. This cycle time includes the broadcasting delay of 6 ns. If 1.2- μ m



CMOS is used, the cycle time will be approximately 33 ns. The chip area of the DPPM engine for a block size of 128 is roughly $200 \times 100 \text{ mil}^2$.

CMOS technology. We can achieve even higher bandwidth by using advanced packaging technology that supports higher I/O pin counts.²²

Figure 6 shows the average number of comparison cycles per block C , measured at different block sizes. Although the pattern characters are serially compared to the data block, early mismatch detection allows the algorithm to search the next block as soon as a mismatch is detected. This feature is especially effective at smaller block sizes where the probabilities of matching the first few characters of the pattern are low. Without early mismatch detection, C equals the average pattern length, in this case 5.88. At larger block sizes, the probability of finding the pattern in the block is higher, thus the value of C also increases.

Using 50 ns as the comparison cycle time T_c , Figure 7 shows the search rates R_b at different block sizes. Recall that increasing the block size requires proportionately more comparators on a chip. At a block size of 16, the search rate is 212 Mbytes/s. This rate matches the predicted optical disk trans-

Figure 7. Search rates at different block sizes.

fer rate of 200 Mbytes/s.¹³ At a block size of 128, the search rate reaches 1 Gbyte/s. This rate is sufficient to handle existing memory bandwidth of supercomputers as well as data input from optical-fiber transmission systems in future communication networks.

Extension and integration. Instead of comparing the pattern characters serially to the data block as in the DPPM algorithm, it is possible to compare multiple pattern characters to the data block. The Multiple-Pattern, Multiple-Data (MPMD) approach reduces the number of comparison cycles required per block and thus results in a higher search rate. Since multiple-pattern characters can be grouped into a word with proper length (for example, four characters long for a 32-bit processor), the MPMD engine can be easily integrated with the relational data filter previously described so that the search operation can be performed on both formatted and unformatted data.

An MPMD engine uses a 2D array of comparators to perform multiple steps of the DPPM algorithm in parallel. Pattern characters can be loaded and compared in parallel to the input data block. The partial match traces propagate through the comparators in a diagonal direction and terminate at the V registers for partial matches and T registers for matches. The whole parallel comparison operation can be implemented in one or more pipeline stages. Notice that the maximum length of the critical path equals the minimum length of the input block and the pattern.

To integrate the MPMD engine in the experimental relational data filter, the designer must provide interfaces to pass the control information between the MPMD and the data filter instruction set. A Match flag can be used to indicate one or more matches terminated at the current input data block (indicated in the T register). This Match flag can be used as a condition code for Boolean predicate evaluation instructions. Furthermore, since all the state information of the string search operation is stored in the V and T registers, these registers can be mapped as a part of the register file. Since the string search operation is performed while the data is loaded into the relational data filter, the search results for each input record can be stored in flags and registers. The text is also segmented into 128-byte records. The results are then forwarded to the relational filtering operation in pipelined fashion. The relational data filter can then use and manipulate directly the search results stored in the condition codes and in the general-purpose registers.

The speed of the MPMD circuit is fast enough to implement in one pipeline stage that matches the VLSI chip I/O rate. The critical path of the MPMD circuit for an input data block size of 4 bytes wide (8 bits per byte) consists of a byte comparator plus four stages of combinational logic for partial match propagation. Using a conservative VLSI technology with about 300-ps gate delay time, the byte comparison will take about 5 ns (including pattern and data broadcast time,

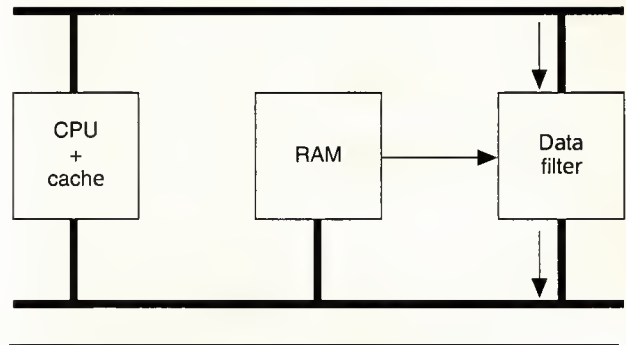


Figure 8. Accelerator and processor integration.

XOR comparison logic, and eight carry propagation stages.) The partial match signal propagating through four stages (4-byte comparator in diagonal) will take about 1.2 ns. Thus, the total time of each parallel comparison will be less than 8 ns. Since this comparison speed matches the maximum data input rate of high-performance I/O pads,²¹ it is unnecessary to introduce extra pipeline stages for the comparator array.

For concurrent query evaluation, we can construct multiple-parallel MPMD building blocks that allow multiple text patterns to be searched simultaneously. To do that, multiple Match flags can be indexed for further Boolean predicate evaluation. With current VLSI technology, we estimate that thirty-two 8-byte-long text patterns can be compared in parallel (total 1,000-byte comparators) in one VLSI chip. This estimation is conservative; however, a small VLSI chip may significantly increase the yield and reduce the cost. Such an accelerator can search 32 text patterns concurrently at about 500-Mbyte/s input data rates.

VLSI accelerators can achieve very fast string searches, at rates easily exceeding the I/O data rate of VLSI chips. The density of current VLSI technology can further improve the throughput of concurrent string searches. The test string matching operations can operate synchronously with the Boolean predicate evaluation operations used in data scan operations.

Application notes

The proposed experimental VLSI accelerators can enhance the performance of general-purpose computing and dedicated systems.

General-purpose computer systems. We use an abstract data processing model to illustrate the applications of the experimental VLSI accelerators for general-purpose computing systems. The abstract model consists of a set of storage devices and a set of VLSI search accelerators interconnected by a high-performance bus interconnection network. This network provides fast virtual circuit connections for communication sessions required by the applications. As shown in Figure 8, designers can easily incorporate the accelerators

into a board-level processing node. A typical processing node consists of a CPU with high-speed caches as well as a large set of dual-ported main memory. The accelerators process data on a page-by-page basis. The data comes through the bus from either the main memory or other nodes. While the data filter is executing the search and aggregation operations, the CPU can execute other tasks. When the search and aggregation operations are completed, the search results can be accessed by the CPU through the main memory.

For a parallel computer, multiple processing nodes are connected by an interconnection network (the design of which is beyond the scope of this article). A search operation starts from the loading of search instructions into one or more data filters. Communication channels between the data sets and the accelerators are established to form a filter network. After the filter network is formed, the data passes through the network in a pipelined fashion. The input and output of each data filter are buffered synchronously or asynchronously, depending on the applications. Figure 9 shows a tree structured associative search task designed to select data records satisfying the predicate ((C and A) or (C and B)), where A, B, C are patterns. The first two accelerators search for two disjunctive patterns A and B, respectively, from two data sets. The third accelerator searches for pattern C from records selected by the first two filters.

Dedicated systems. The experimental VLSI data filter is a critical component in a large-scale broadcasting database system prototype called the Databycle architecture.¹⁷ The Databycle architecture demonstrates the feasibility of providing a high-performance and highly flexible database access services for a large number of users in geographically distributed areas. Databycle consists of a set of data and data filters located in multiple distributed sites. The system broadcasts data to data filters cyclically through optical fibers. (To support applications with a very large volume of data, only the

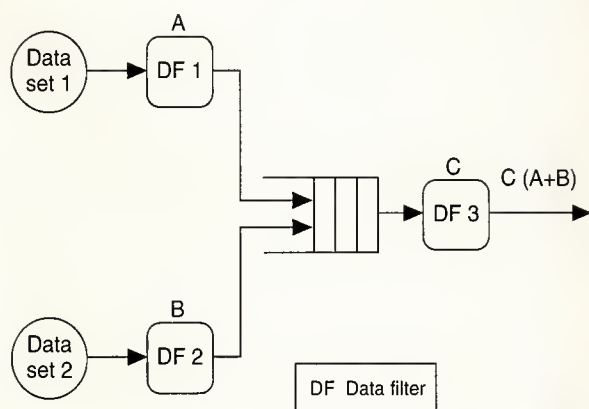


Figure 9. A filter network.

part of the data that is constantly changing or frequently accessed by multiple sites are broadcast repetitively.) Search queries are translated into data filter instructions and distributed to multiple data filters that execute them concurrently.

Since the data is made available to all the data filters simultaneously, the system can concurrently process an unlimited number of queries. This broadcast property is fundamental in supporting concurrent associative search, aggregation, and transaction processing services for millions of users on a set of constantly changing data.

To demonstrate its implementation feasibility, we installed a prototype system of the Databycle architecture in our laboratory. It consists of silicon memory-based data storage devices, 500-Mbps fiber-optic broadcasting links, and multiple formatted data filter boards. The architecture of a distributed site containing data sets is similar to the general-purpose node architecture shown in Figure 8. A distributed site that performs only data accessing contains three VLSI data filters and associated controllers. The prototype system demonstrated the functionality of the Databycle architecture. An extended query set can be programmed with the data filter instructions. These extended queries execute in two phases. First, the data filter prefilters to reduce the amount of data to be processed later. Second, the general-purpose CPU completes the query processing and retrieves the data.

We learned several lessons from the Databycle prototype. First, the effort involved in the VLSI accelerator design is very small compared to the total effort involved in software development and system integration (less than 20 percent). This is partly due to the advancement in CAD tools but mostly due to the increasing complexity of system and application software. Second, the use of VLSI accelerators significantly improves the performance of the overall system and enables new capabilities more economically.

Frieder, Lee, and Mak proposed a document-searching architecture, based on the DPPM engines, to increase the throughput of an information retrieval system.^{23,24} The proposed architecture (Figure 10 on page 18) is comprised of a set of DPPM engines and a single master processing element (PE). An information retrieval query is decomposed by the PE into basic match primitives to be executed in the DPPM engines, while the query (a sequence of operators) is evaluated at the PE using results from the document-search engines. By separating the operator and query complexity from the DPPM engines, the complexity of the customized hardware is freed from the complexity of the query.

The PE instruction set is based on the text retrieval machine instruction set presented by Hollaar et al.²⁵ but modified to be match-based. In other words, each instruction is defined as a set of match conditions with a simple imposed control structure. The instruction set²³ supports Boolean operations of patterns as well as wild-card characters that match one or more characters in the pattern. As the imposed

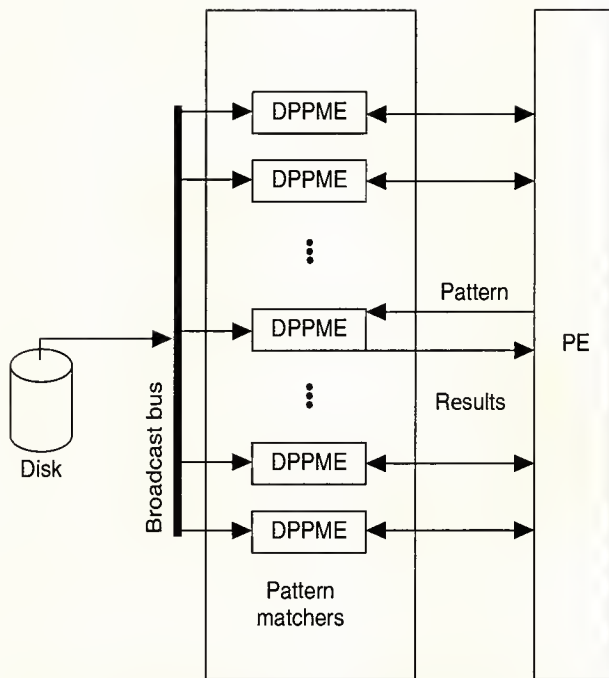


Figure 10. Proposed document-searching architecture.

control structure requires minimal computation time relative to the amount of processing involved in searching the document stream, it can be implemented in software without significantly affecting the system throughput. A 50-MIPS PE can support 10 DPPM engines of 128 blocks in parallel, at a utilization level of 0.25, with each DPPM engine having a search rate of 1 Gbyte/s. At this search rate, we can search both the Old and the New Testaments of the Bible in 5 ms, *Webster's Dictionary* in 16 ms, and the entire volume of *Encyclopaedia Britannica* in 400 ms.

Related work

Numerous hardware-based solutions have been proposed and implemented to expedite both the associative and string search operations. As fast software string-searching algorithms^{26,27} are based on finite-state automata (FSA), hardware realizations of FSA pattern matching were investigated by several researchers.²⁸⁻³⁰ FSA requires precompilation of the patterns and processes the data string one character at a time. Although precompilation of the pattern eliminates the need to compare each character of the data string to every pattern character, the sequential character-at-a-time processing severely limits the search rates of these systems.

Several researchers³¹⁻³⁴ propose using comparator arrays to perform pipelined pattern matching directly without precompilation of the patterns. Multiple patterns are com-

pared concurrently to the data string to achieve higher throughput. However, the search rate is still limited by the sequential processing of the data string. In the systolic array approach,³¹ data and pattern characters are routed in opposite directions. At any given clock cycle, only half of the cells in the array can perform meaningful computation. Therefore, half of the physical hardware is actually wasted. In some proposals,^{32,34} pattern characters are preloaded into the comparator cells. Each character of the data string is broadcast into all cells serially, and comparison results are generated by all cells simultaneously. Since a string search operation on text database exhibits very low selectivity, comparisons beyond the first few characters of the pattern are usually unnecessary. Thus, most of the comparisons with the last few characters of the pattern are redundant.

To reduce the number of redundant comparisons and to increase the degree of effective parallelism in the pattern matching problem, both Altep³⁵ and our string-search algorithm use a data parallel, pattern serial scheme in which pattern characters are broadcast and compared to a block of the data string in parallel. While Altep is a cellular processor optimized for regular expression comparisons with microprogrammed control, our DPPM engine is a VLSI filter optimized for variable-length text processing with hard-wired control. The decoupling of query resolution from the primitive match operation simplifies the structure of the DPPM engine so that it can be implemented compactly and is hence more efficient.

Relational data filtering is different from unformatted string searching in that in the former, the starting position of the pattern to be matched is specified in the record format definition. As a result, associative search on relational data requires fewer comparison operations than string search. Another difference is that relational data filtering requires more complex search predicates than does unformatted string searching. Therefore, the filters designed for string search are not efficient for relational data filtering.

The proposed relational data filter differs from other filters^{28,29,36-39} in that it is a RISC architecture with a highly programmable instruction set and efficient implementation. As indicated earlier, the instruction set of the data filter can construct a wide mixture of associative search and aggregation queries based on efficient Boolean predicate and conditional assignment operations. The simplicity of the reduced instructions also results in fast VLSI implementation.

THE PROPOSED DATABASE ACCELERATOR can offload the CPU from the time- and space-consuming associative search and aggregation operations. The novel instruction set and Boolean accumulator support allow highly pipelined Boolean predicate evaluation for on-the-fly processing over high-speed input data streams available in the future storage technology. The efficient VLSI architecture results in a 64-

Mbyte/s associative search rate via a simple design with modest fabrication technology. The architecture can be easily extended to support a higher data rate by increasing the width of the data path and the number of pipeline stages. The VLSI data filter achieves significant performance advantages for synchronous searches of formatted data when compared with general-purpose microprocessors. Its efficiency increases when we integrate several subsystems onto a single VLSI chip.

The VLSI DPPM engine overcomes the fundamental limitation of sequential pattern-matching algorithms and efficiently processes consecutive text strings in variable block size. We can achieve a search rate of Gbytes/s with simple VLSI implementation of the algorithm. By incorporating the DPPM engine with high-speed data channels available in silicon memory and optical storage devices, we can economically resolve the problem of slow response time for large information service databases.

Through the actual design and the use of VLSI accelerators, we learned that VLSI accelerators are a very cost-effective means for improving overall system performance. Modern VLSI design tools can significantly reduce the time and the cost believed to be required for custom IC design. As a consequence, the VLSI accelerator approach is a relatively low-risk and inexpensive one when the overall software and hardware system design and integration costs are considered. Additionally, we learned that the value of VLSI accelerator design is not limited to the system performance gain. Basic functional components and I/O structures abstracted from the accelerator designs can potentially become standard features for future general-purpose CPU architecture designs. ■

Acknowledgments

We acknowledge Ophir Frieder for his contribution to various aspects of the DPPM work. We also acknowledge Tony McAuley, Yow-Jian Lin, Jane Cameron, and Bill Mansfield for their valuable comments, which significantly improved the quality of this article.

References

1. C.K. Baru and S.Y.W. Su, "The Architecture of SM3: A Dynamically Partitionable Multicomputer System," *IEEE Trans. Computers*, Vol. C-35, No. 9, Sept. 1986, pp. 790-802.
2. D.K. Hsiao, ed., *Advanced Database Machine Architectures*, Prentice Hall, Englewood Cliffs, N.J., 1983, pp. 1-18.
3. W. Kim, D. Gajski, and D. Kuck, "A Parallel Pipelined Relational Query Processor," *ACM Trans. Database Systems*, Vol. 9, No. 2, June 1984, pp. 214-242.
4. M. Kitsuregawa, M. Nakano, and M. Takagi, "Query Execution for Large Relations on Functional Disk System," *Proc. Fifth Int'l Conf. Data Engineering*, IEEE Computer Society Press, Los Alamitos, Calif., 1988, pp. 159-167.
5. E. Ozkarahan, *Database Machine and Database Management*, Prentice Hall, Vol. 10, 1986, pp. 319-337.
6. *DBC/1012 Data Base Computer: Concepts and Facilities*, C02-0001-05 Release 3.1, Teradata, Inc., 1989.
7. C.K. Baru and O. Frieder, "Database Operations in a Cube-Connected Multicomputer System," *IEEE Trans. Computers*, Vol. 38, No. 6, June 1989, pp. 920-927.
8. D. DeWitt et al., "GAMMA—A High Performance Dataflow Database Machine," *Proc. VLDB Conf.*, Morgan Kaufmann Publishers, Los Altos, Calif., 1986, pp. 228-237.
9. M. Smith et al., "An Experiment on Response Time Scalability in Bubba," *Proc. Int'l Workshop on Database Machines*, Springer-Verlag, New York, 1989, pp. 34-57.
10. M. Kitsuregawa et al., "Design and Implementation of High Speed Pipeline Merge Sorter With Run Length Tuning Mechanism," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, Boston, 1988, pp. 89-102.
11. C. Lee, S.Y. Su, and H. Lam, "Algorithms for Sorting and Sort-Based Database Operations Using a Special-Function Unit," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic Publishers, 1988, pp. 103-116.
12. T. Matoba et al., "A Rapid Turnaround Design of a High Speed VLSI Search Processor," *Integration, VLSI J.*, Vol. 10, 1991, pp. 319-337.
13. P.B. Berra and N.B. Troullinos, "Optical Techniques and Data/Knowledge Base Machines," *Computer*, Vol. 20, No. 10, Oct. 1987, pp. 59-70.
14. S. Redfield and L. Hesselink, "Data Storage in Photorefractives Revisited," *Proc. Conf. on Optical Computing*, Vol. 963, 1988, pp. 35-45.
15. T.A. Shull, R.M. Holloway, and B.A. Conway, "NASA Spaceborne Optical Disk Recorder Development," *Optical Storage Technology and Applications*, Vol. 899, 1988, pp. 272-278.
16. H. Garcia-Molina, R.J. Lipton, and J. Valdes, "A Massive Memory Machine," *IEEE Trans. Computers*, Vol. C-23, No. 5, May 1984, pp. 391-399.
17. G.E. Herman et al., "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM-SIGMOD Conf.*, ACM, N.Y., May 1987, pp. 97-103.
18. J. Hennessy, "VLSI Processor Architecture," *IEEE Trans. Computers*, Vol. C-33, No. 12, Dec. 1984, pp. 1221-1245.
19. S.A. Przybylski et al., "Organization and Implementation of MIPS," in *Tutorial on Advanced Microprocessors and High-Level Language Computer Architecture*, V. Milutinovic, ed., IEEE CS Press, 1984, pp. 202-239.
20. C.J. Date, *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., 1986.
21. W. Marcus, "A CMOS Batchier and Banyan Chip Set for B-ISDN

- Packet Switching," *IEEE J. Solid-State Circuits*, Vol. 25, No. 6, 1990, pp.1426-1432.
22. R. Johnson, "Multichip Modules: Next-Generation Packages," *IEEE Spectrum*, Vol. 27, No. 3, Mar. 1990, pp. 34-48.
 23. V.W. Mak, K.C. Lee, and O. Frieder, "Exploiting Parallelism in Pattern Matching: An Information Retrieval Application," *ACM Trans. Information Systems*, Vol. 9, No. 1, Jan. 1991, pp. 52-74.
 24. O. Frieder, K.C. Lee, and V.W. Mak, "JAS: A Parallel VLSI Architecture for Text Processing," *Data Engineering*, Vol. 12, No. 1, Mar. 1989, pp. 16-22.
 25. L.A. Hollaar et al., "Architecture and Operation of a Large, Full-Text Information Retrieval System," in *Advanced Database Machine Architecture*, D.K. Hsiao, ed., Prentice Hall, 1983, pp. 256-299.
 26. R.S. Boyer and J. Moore, "A Fast String Searching Algorithm," *Comm. ACM*, Vol. 20, Oct. 1977, pp. 762-772.
 27. D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Computing*, Vol. 6, June 1977, pp. 323-350.
 28. F. Bancilhon et al., "Verso: A Relational Backend Database Machine," in *Advanced Database Machine Architectures*, D.K. Hsiao, ed., Prentice Hall, 1983, pp. 1-18.
 29. R. Gonzalez-Rubio, J. Rohmer, and D. Terral, "The Schuss Filter: A Processor for Nonnumerical Data Processing," *Proc. ACM SIGARCH Conf.*, 1984, pp 64-73.
 30. R.L. Haskin and L.A. Hollaar, "Operational Characteristics of a Hardware-Based Pattern Matcher," *ACM Trans. Database Systems*, Vol. 8, Mar. 1983, pp. 15-40.
 31. M.J. Foster and H.T. Kung, "The Design of Special Purpose Chips," *Computer*, Vol. 13, No. 1, Jan. 1980, pp. 26-40.
 32. A. Halaas, "A Systolic VLSI Matrix for a Family of Fundamental Search Problems," *Integration, VLSI J.*, Vol. 1, Dec. 1983, pp. 269-282.
 33. H.T. Kung and P.L. Lehman, "Systolic VLSI Array for Relational Database Operations," *Proc. ACM SIGMOD Conf.*, May 1980.
 34. K. Takahashi, H. Yamada, and M. Hirata, "Intelligent String Search Processor to Accelerate Text Information Retrieval," *Proc. Fifth Int'l Workshop on Database Machines*, Kluwer Academic Publishers, Oct. 1987, pp. 440-453.
 35. D. Lee, "Altep—A Cellular Processor for High-Speed Pattern Matching," *New Generation Computing*, Vol. 4, Sept. 1986, pp. 225-244.
 36. P. Faudemay and P. Valduriez, "Design and Analysis of a Direct Filter Using Parallel Comparators," *The Fourth Int'l Workshop on Database Machines*, D.J. DeWitt and H. Boral, eds., Springer-Verlag, Mar. 1985.
 37. G. Gardarin et al., "SABRE: A Relational Database System for a Multimicroprocessor Machine," D.K. Hsiao, ed., *Advanced Database Machine Architectures*, Prentice Hall, 1983, pp. 19-35.
 38. H. Schweppe et al., "RDBM—A Dedicated Multiprocessor System for Database Management," D.K. Hsiao, ed., *Advanced Database Machine Architectures*, Prentice Hall, 1983, pp. 36-85.
 39. S.Y.W. Su et al., "The Architecture Features and Implementation Techniques of the Multicell CASSM," *IEEE Trans. Computers*, Vol. C-28, No. 6, June 1979, pp. 430-445.



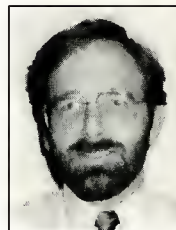
Kuo Chu Lee is a researcher in the Applied Research Area of Bellcore in Morristown, New Jersey. His interests include distributed cooperative systems, extended transaction management systems, and design of large-scale multimedia systems. Lee received a BS in power mechanical engineering from National Tsing-Hua University, an MS in electrical engineering from Ohio State University, and a PhD in computer engineering from Rutgers University. He is a member of the IEEE.



Takako Matoba Hickey is a researcher in the same area. Her research interests include fault-tolerant computing, distributed systems, and software engineering. She received a BS degree in engineering and applied science from California Institute of Technology and an MS in computer science from Stanford University.



Victor W. Mak is also a researcher in the Applied Research Area of Bellcore. His research interests include distributed systems, parallel processing, object-oriented programming, and performance evaluation. Mak received a BS degree in electrical engineering from the University of Toronto and MS and PhD degrees in electrical engineering from Stanford University. He is a member of the IEEE and ACM.



Gary E. Herman is division manager of Network Systems Research in the Applied Research Area of Bellcore. He received the BSE and PhD degrees in electrical engineering from Duke University.

Direct questions concerning this article to K.C. Lee at Bellcore, 445 South Street, Morristown, NJ 07962-1910; or via e-mail at kasey@thumper.bellcore.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153

Medium 154

High 155



An Associative Accelerator for Large Databases

The RAPID-1 (Relational Access Processor for Intelligent Data) is an associative accelerator that recognizes tuples and logical formulas. It evaluates logical formulas instantiated by the current tuple, or record, and operates on whole relations or on hashing buckets. RAPID-1 uses a reduced instruction set and hardwired control and executes all comparisons in a bit-parallel mode. It speeds up the database by a significant factor and will adapt to future generations of microprocessors.

Pascal Faudemay

Mongia Mhiri

Laboratoire MASI

Information servers should perform at a higher level to be usable for more numerous applications. Among such applications are real-time information systems; large knowledge bases; and very large databases, such as those that hold seismic, textual, and financial data. A solution to the performance problem may come from better algorithms, more parallel systems, or more calculating power in the processor. In any case, good processing power is desirable and may reduce the complexity of parallel machines or help to improve performance on sequential machines.

To increase the power of general processors, we can rely on processors themselves, or on application-specific integrated circuit (ASIC) accelerators. The old reasoning that specific circuits, which were not yet ASICs, cannot perform as well as general circuits contradicts all research on ASIC and ASIC-dedicated CAD.¹⁻³ Today's ASIC circuits are designed at the schema level, using standard cells, and the layout can be shrunk or changed automatically, according to the technology. Specific architectures have been developed for information servers,^{4,5} and most of them use specific circuits. Recent machines deliver very high performance.⁶ Among the various accelerator types,⁶⁻¹¹ associative circuits¹²⁻¹⁴ seem to offer a natural solution for extended relational servers (see box on information servers) and for knowledge base servers.

Information servers retrieve and update data,

based on assertions of the data to be retrieved. Associative circuits also retrieve in parallel a set of data, based on a description of its content. Therefore, associative circuits seem well-adapted to the operations of extended relational servers and knowledge bases, as well as set-oriented operations on object-oriented information servers.

Associative circuits use Cartesian product algorithms, in which operations on sets of tuples (relations) are done by considering all pairs of tuples. Each cell or processing element of the associative circuit compares a tuple, or part of a tuple, which is memorized in this cell with the comparand tuple. (See box on associative circuits.) It is more efficient to join in parallel two relations by Cartesian product than to sort-join in parallel when the number of processing elements is a significant fraction of the number of tuples of one of the relations.¹⁵

The best way to satisfy this condition is to divide the operations into smaller ones compatible with the size of the associative circuits. The most efficient solutions are hash-Cartesian product algorithms. Comparisons between tuples belonging to two source relations are limited to comparisons of hashing buckets having the same index.¹⁶ Therefore, the desirable number of processors is limited to the number of tuples in the hashing bucket of the smaller relation, multiplied by the number of processing elements used for each tuple (usually one PE per tuple).

Hash-product solutions can be implemented

Information servers

Information servers manage databases in which elementary data may belong to any type and knowledge bases (databases of rules) for the benefit of users and applications distributed on an interconnection network. Database systems with the best performances for the most-used functions are extended relational systems. These systems rely on the relational data model, which defines unary and binary operations on relations. Relations are subsets of Cartesian products of domains, such as integers or real numbers. An element of a relation is a tuple, which can be represented as a record. Tuples are composed of attributes (possibly implemented as fields in a record) that take their value in the domains which define the relation.

The operations of the relational model are

- **SELECTION**, which returns the tuples that satisfy a logical formula;
- **PROJECTION**, which eliminates duplicate tuples resulting from the suppression of some attributes;
- **JOIN**, which is a selection on the Cartesian product of two relations; and
- **SET OPERATIONS** on two relations (union, intersection, and so on.)

Relational languages also include sorting, aggregates with or without grouped-by (for example, the average of a sal-

ary grouped by region), and sometimes the transitive closure on a relation. The RAPID circuit implements all these operations, though the transitive closure is not part of its instruction set.

Extended relational models define their attributes on domains corresponding to abstract data types defined by the user. They possibly can represent polygons or images. Tuples and attributes may be of any size up to several Mbytes. RAPID processes abstract data types in a simple way.

Other data models that are presently developed are object-oriented models in which methods are defined for each type of persistent or nonpersistent object and used for accessing or updating them. These object-oriented data models usually rely on object graphs in which links are represented by constant identifiers, not by pointers. These models are less advanced in terms of performance, therefore less adapted to the implementation of such servers as real-time information servers. However, they are used for an internal representation of objects in extended relational systems, which can use object managers for access and updates to objects. The software kernel of the RAPID machine uses the services of an object manager. On the other hand, object graphs can always be represented through the relation super-type, especially to execute operations on these graphs in a set-oriented manner.

by using a software hashing before the Cartesian product or by hashing the stored tuples and tuples from the second source relation by hardware at storage time and evaluation time, as it is done in some set-associative memories. In this case, however, one of the source relations must more or less remain in the associative circuit, even if the number of processing elements is smaller than the number of tuples.^{17,18}

We chose to use algorithms based on the hash-product, with a software implementation of hashing. Below, we define our circuit design goals, the data structures that the circuit recognizes, the instruction set and associated methods, and the architecture and performances. Finally, we show how this type of circuit may correspond to the needs of several generations of microprocessors.

Main issues

Today's associative circuits do not seem fully adapted to general database queries. They are either too specific or too slow. In addition, a main issue in the design of an associative circuit for databases is whether to use a small associative accelerator, which can be added to any machine, or imple-

Associative circuits

Associative circuits execute retrievals and updates on data based on

- their content,
- an assertion on this content, and, most often,
- on a comparand, or key data, which is compared to data stored into the circuit, and used to instantiate the assertion.

They are mainly composed of an array of associative cells, or processing elements, that execute comparisons in parallel, and of one or several functional blocks that use the comparison results in order to solve the query. These circuits can also execute other operations, such as sorting, proximity evaluations on the comparand and data (for example, Hamming distance), or other operations.

Associative memory capacity

Associative memories may be fully associative memories or set-associative memories. In the latter, data are distributed between sets by a hashing function, and a comparand compares to memorized data only within the set with the same hashing value.

Fully associative memories with the most advanced memory capacity offer 20 Kbits per component, which is more than 500 words with 32 bits width for each word.²² Solutions with the best integration density use five transistors per 2-bit cell, that is, 2.5 transistors per bit.²³ If used to compare tuples belonging to a couple of hashing buckets, the corresponding size of these hashing buckets is very satisfactory and is not even necessary for high efficiency. However, with this circuit type, data cannot remain in the circuit and must be transferred into it before the operation.

On the other hand, if the circuit is to be used as a data cache, it needs a bigger memory. Usual relation sizes are between 100,000 and 1 million tuples, with current tuple sizes of 100 bytes or more. Database sizes range from one Gbyte to terabytes for seismic applications or satellite data. The solution may be a set-associative memory.

With set-associative memories, we can separate logic and memory, which may be a classic RAM. We can then achieve large memory capacities, with approximatively twice the same component count and price. However, the memory capacities that are presently considered are not much larger than 10 Mbytes, which is much smaller than database sizes. The slow-down due to disk access limits the device speedup to about a factor of two (end-of-line).^{17,24} Set-associative memories have also been integrated on a single chip with memory capacities up to 160 Kbits per chip.³⁴

The hashing function depends on the operation. It may therefore require reorganizing the data before an operation, which may bring us back to the situation where data are transferred from memory. Retrieval through set-associative operations in an information server is therefore quite different from the usual use of set-associative caches, where retrieval is based on the logical addresses of data.

Due to the limited size of associative caches and needs for rehashing, smaller associative memories with non-resident data may presently be a good solution.

and frequent queries are based on joins, aggregates, sorting, and text retrieval. The three queries are implemented only by specialized associative memories.¹⁹⁻²¹ Therefore, we wanted to implement these four operations in a single associative processor. We chose to implement all query operations in the same hardware to simplify the software.

We also felt that the length of the tuple must be arbitrary. In several existing circuits, the length of a record is limited—for example, 32 bytes in Ogura's²² and 256 bytes in the Advanced Micro Devices 85C95. Larger tuples may be processed by using several components, but the record length is often a characteristic of the machine. In RAPID-1, tuples may have any length within the total memory capacity of the circuit, which may also use several components. The circuit is optimized for a useful length of the tuple (attributes taking part in the comparisons) between 4 and 100 bytes. As in Ogura's circuit and in the Data Base Accelerator,²³ the reduction of the useful tuple length may increase the number of simultaneously evaluated tuples.

Memory capacity. Considering the present limitations of associative and set-associative cache solutions, we chose to design a fully associative circuit with a small amount of memory. The general processor transfers data during all operations. Performance depends on software efficiency and on the power of the host processor. In this "co-architecture," performance increases with the power of the microprocessor, up to the saturation of the associative circuit. This arrangement is therefore well adapted to the fast evolution of general processors. (See box on associative memory capacity.)

Performance. Classic associative circuits cycle in less than 100 ns, but a comparison set implies several instructions. Comparisons using an equality criterium operate in a wide-bandwidth manner (bits in parallel mode), but comparisons on inequality criteria use serial calculations on each successive bit of the comparand (serial-bit mode). The duration of a one-word comparison is thus usually 32 cycles or more. This difference in performance between equality and inequality comparisons will imply a circuit inefficiency for inequality comparisons.

We implemented bit-parallel comparisons for all operations, which means the use of a comparator in each cell. This solution is costly in terms of circuit area, and reduces the number of tuple comparisons that can be done in parallel, but it guarantees good performance in all cases. We also used a reduced instruction set of high-level instructions, each executed in one cycle, to contribute to this purpose.

Data structures

The RAPID circuit^{25,26} recognizes two types of data structures: tuples and logical formulas. Data are memorized in the circuit as logical formulas while keys are broadcast to the processing elements as tuples. We shall explain here the

ment a complete associative machine based on an associative cache.

Functions. According to the applications, the most costly

mapping between tuples and logical formulas.

Tuples are variable-length structures, with an arbitrary number of attributes, or fields, each with an arbitrary length. The only limitation to the attribute or tuple length is the total memory size of the circuit, which may be composed of an arbitrary number of components. The internal data representation is the byte string. The comparison between numbers in two complements is mapped into a byte-string comparison by changing the appropriate bit. This operation is presently done by software. A similar approach is used for real numbers.

Each tuple is referenced by a 32-bit identifier, which may be a pointer, a link (a pointer on a pointer), or an identifier that does not change when the tuple is updated. The tuple is stored in the circuit as a logical formula, as we describe later. The tuple identifier then becomes the identifier of the corresponding subexpression. This identifier is then stored in a specific register of each cell participating in the subexpression evaluation. When the circuit compares a key tuple with a set of tuples stored as subexpressions, the result may be the sequence of identifiers of the subexpressions that are satisfied after their instantiation by the key tuple.

The circuit also recognizes logical formulas. These are composed of subexpressions that are, in turn, composed of predicates. Predicates are connected by the functions And or Or, according to the priority used in the logical formula, or by the special function Then (a variant of And), that we will explain later. The complementary function connects subexpressions. The circuit can either evaluate the global value of the logical formula or evaluate in parallel each subexpression. The result is either the set of identifiers of satisfied subexpressions or the quantity of these subexpressions. The global (Boolean) value of the formula is still available in any case. The evaluation of a fuzzy logical formula is possible, through the return of the weight of each satisfied subexpression. Such fuzzy, or vectorial, evaluation can rank documents or paragraphs by value.^{27,28} The circuit can also take into account (or not) the order of retrieved words in a sentence or document.

Predicates are either attribute-operator-constant, or attribute-operator-attribute. Each one is usually evaluated by a distinct processing element. When an argument is an attribute, it is represented by an attribute number stored in a specific location of the local memory of the processing element (PE). When a PE must compare an attribute with a constant, it waits for the arrival of the attribute number of the comparison, which usually follows the end-of-attribute (FA) signal corresponding to the previous attribute or the new-tuple (C3) signal. If the information on the data bus is equal to the predicate attribute number, the PE begins an evaluation phase. If not, the PE remains idle until the next attribute number arrives. The proportion of PEs used in parallel varies according to the query; however, in most operations it is 100 percent.

Comparison operators include $<$, \leq , $>$, \geq , $=$, \neq , Included in, Contains, Strictly included, and Strictly contains. The predicate "arg1 Included in arg2" corresponds to the search for a nonanchored byte string within another byte string. Text attributes can be divided into words that are separated by punctuation characters defined by the user. Inclusion and comparison predicates may be completed by the prefix information, which means that the retrieval is anchored at the beginning of an attribute or a word but may end before the end of the word.

There is also suffix information, corresponding to the fact that the character string does not necessarily begin at the beginning of an attribute or word, but that it ends at an attribute or word end. Specific word information indicates whether word separations are taken into account for a given predicate.

Subexpressions are defined by

subexp = predicate | subexp connect predicate, with
connect = And | Or | Then.

Then is a variant of And. It corresponds to one of the following:

- a succession of nonanchored retrievals in the same attribute;
- the comparison of tuples according to a succession of sorting criteria; or
- the successive comparison of fractions of an attribute and of a large (more than 32 bytes) constant.

In some cases, comparisons are made on calculated results. As an example, we can retrieve the names of people who earn at least 1.1 times more than the average salary of their region. (See sidebar on associative memory capacity.) In this case, the software evaluates a virtual attribute, which receives a number (for example, attribute 4), such that, Attribute 4 = $1.1 \times$ Attribute 3. This attribute is stored as a predicate constant and/or broadcast as a key in place of an ordinary attribute.

Instruction set

Figure 1 (on page 26) gives the instruction set of the RAPID-1 component. Instructions are stored in a codop (operation code) field and specified by an operator code, Oper, and by complementary bits. One instruction word is associated with each predicate, or more generally, to each PE. The circuit is controlled by data. No instruction is needed at evaluation time.

A control word (10 bits) broadcasts with every data word (16 data bits, plus 6 special bits).²⁵ Therefore, the writing of data into the circuit makes full use of the usual 32-bit word width.

SELECT
JOIN
PROJECTION
SORTING
SEMI-JOIN
SET OPERATIONS
NOP
RESERVED

Figure 1. Instruction set.

This instruction set should evolve in the next version by the suppression of the SEMI-JOIN and SET OPERATIONS instructions, which perform the same function as selection. Two important operations, EXTERNAL SORT and AGGREGATE with GROUPED BY, are executed by using the JOIN and PROJECTION operations. We briefly present the first four instructions.

SELECT. This two-phase operation loads and evaluates the selection expression. It returns a Boolean result for each evaluated tuple. The circuit overflow is not presently admitted. The SELECT query must then be decomposed into several queries.

JOIN. This is also a two-phased operation that loads tuples from one hashing bucket as a sequence of logical subexpressions, and evaluates the other hashing bucket. Predicates load sequentially. After a predicate has been stored in a PE, the PE transmits control to the next PE. For each key (which belongs to the second hashing bucket), the operation returns the sequence of tuple identifiers of the first hashing bucket that satisfies the JOIN condition. At the end of each tuple evaluation, to inform the PEs that none of them should wait to write its identifier on the bus, a specific C3 signal ends the identifiers' write phase. This signal can also interrupt the transmission of the identifier sequence.

This last mechanism is used in the EXTERNAL (distributed) SORT operation, which uses the JOIN instruction. The results of the external sort operation are the identifiers of the sorting limits. He et al. studied efficient methods for the choice of limits.²⁹

PROJECTION. The projection operation is a single load-evaluation phase. The new tuple from the current hashing bucket loads into the first free PE, and is compared at the same time with tuples loaded as a logical formula in other PEs. The representation of a set of tuples as a logical formula for a projection is given in Figure 2, where $(a(i), \dots, a(i+n))$, $(b(i), \dots, b(i+n))$, and so on, are the values of tuples that are not duplicates. If the value of the global logical formula is *false*, the current tuple is not a duplicate. In this case the software kernel stores the tuple identifier in the result relation. The control for the loading of the next tuple transfers to the next free PE. If the value of the logical formula is *true*, the

Att i = a(i)	and
...	
Att i+n = (i+n)	or
Att i = b i and	
...	
Att i+n = b (i+n)	or
...	

Figure 2. A set of tuples as a logical formula.

key tuple is a duplicate. The identifier of the logical subexpression satisfied by the key tuple is returned by the circuit. This result is mainly used when using the PROJECTION operation for the calculation of an AGGREGATE with GROUPED BY. The end-of-formula token is not moved, and the PEs used to store the previous tuple are reused for the next one. This operation is fully managed by the sequencers of the relevant PEs, without using an instruction.

These mechanisms can also be used to execute AGGREGATE with GROUPED BY operations, such as the average salary per region in the *people (name, region, salary)* relation. (See box on JOIN example.) To calculate this aggregate, we build an array in which each line corresponds to a class. Columns describe the name of the class (here the region name), the intermediate calculation of the class (here the class cardinality and the total amount of salaries), then the final result for the class. Classes are created when new tuples are evaluated and correspond to new classes. Therefore the array is not ordered by class names. A sort on this column or any other may of course be done at the end of the operation; it is generally a small one. If the class is defined by an interval, the class number is calculated during a virtual attribute calculation step, before the aggregate operation itself. This solution makes possible a fast aggregate execution (due to the efficiency of hashing) and an economical one (no specific instruction). However, it increases—in a limited way—the projection cost.

INTERNAL SORT. The arguments of INTERNAL SORT are a set of tuples (small enough to fit into the accelerator) and a specific qualification expression (a list of attributes of ascending or descending sorts). The operation takes place in two phases: a load phase, in which the set of tuples is loaded into the circuit as a logical formula, and an evaluation phase, in which each tuple of the same set is successively used as a comparand. (See Figure 3.) For each comparand tuple, the associative circuit returns the tuple rank versus the other tuples of the set. The operation result is a rank array that is very comparable to that of the SORT operation in APL,³⁰ though it can include *ex aequo*, that is to say, tuples with the same rank. It is easy to use it to get a rank array without *ex aequo*, then physically reorder the tuples in sort order, but this is not

JOIN example with the RAPID circuit

Let us assume we have relations People (name, region, salary) and Region (name, average salary). The second relation may result from an aggregate operation (with a grouped by clause) on the first one. We shall retrieve the people who earn more than the average salary of their region, by executing a join between those two relations with the qualification expression

People.Region = Region.name and
People.salary \geq Region.average salary

Let us consider the following data:

People	#	name	region	salary
	1	Dupond	Paris	10
	2	Durand	Paris	20
	3	Anatole	Caen	20

Region	#	name	average salary
	1	Paris	15
	2	Caen	20

These two relations may correspond to a hashing bucket with the same number of larger relations, with a hashing function on People.Region and Region.name.

Let us assume we store the tuples from relation Region

in the circuit as logical formulas. The qualification expression will map them into the following logical formula, where subexpressions #1 and #2 will be evaluated in parallel:

PE1	ATT2 = Paris	And	#1
PE2	ATT3 \geq 15	Or	#1
PE3	ATT2 = Caen	And	#2
PE4	ATT3 \geq 20	Or	#2

Attribute numbers stored into the predicates are those of the corresponding attributes of the people relation. Virtual attribute # is the tuple identifier. Subexpression #1 corresponds to tuple #1 from region. Connector Or is mainly used as a separator between subexpressions.

In this example, tuple #1 from People, (Dupond, Paris, 10) does not satisfy the logical formula. Tuple #2 (Durand, Paris, 20) satisfies the subexpression #1 and therefore satisfies the logical formula. The result, which is then delivered by the circuit, is the following set of logical addresses: $E = \{\#1, 0\}$, where identifier 0 means "end of result." In the future, a single bit of the identifier should be used to signal the end of result while minimizing transfers. In the same way the result for tuple #3 is $E = \{\#2, 0\}$. Data broadcast to the circuit are limited to the attributes that participate in the comparison (here, attributes 2 and 3).

```

class_number = 1
While the bucket is not empty
do
  read the class_attribute of the current tuple
  store identifier = class number
  store the predicate <ATTRIBUTE = class_attribute>
  if result identifier = 0
    then begin
      store tuple identifier in result relation
      do aggregate calculus on class = class_number
        /*e.g., increment class cardinality
      class_number = class_number + 1
      move token in the PEs array
    else
      send signal C3 ("send no more identifiers")
      do aggregate calculus on class = identifier
    end
  end
end

```

Figure 3. Aggregate algorithm.

always needed. It may be sufficient to access some elements of each tuple in sort order.

The tuple sets used for internal sorting are buckets resulting from a distributive sort. From the rank array of each bucket, it is usually necessary to calculate the ranks array versus the whole relation. This is done by adding a constant to the ranks of each bucket.

During the loading phase of each sort bucket, sort attributes are stored in the circuit as predicates (for example, as "attribute < constant" in the case of an ascending sort attribute). Successive sort predicates are connected by a Then connector. If successive sort attributes correspond to the same sorting sense, it is possible to compact them in the same PE, in the limits of the memory capacity of the PE. During the evaluation phase, the rank calculation is a count of the number of satisfied subexpressions. In the case of an ascending sort, the number of satisfied expressions would be the number of inferior tuples in sort order. He et al. simulated parameters for an optimization of external and internal sort using the RAPID circuit.²⁹

Architecture

The RAPID accelerator (Figure 4) is an associative circuit that interfaces with the host machine as a memory. Figure 5 shows it is composed of three main functional blocks: the array of associative cells or PEs, the subexpressions resolution block, and the query resolution block. In contrast with other associative circuits such as Ogura's and the Data Base Accelerator, it does not have a bit-management block, because it does not execute bit serial operations. The subexpression resolution block is a specific logical block that executes Ands, Ors, or Thens between predicates. Its operation is detailed later. Most associative circuits do not include this type of functional block, but Ogura et al.²² proposed a functionally similar structure, and Penazola and Ozkarahan³¹ proposed a seemingly similar one. The query resolution module includes the And block, the Or block, the counting block ("+" block), and the arbitration block.

The arbitration block is the classic multiple-match resolution circuit, which is a characteristic of associative memories. (See Figure 6.) As this block is mainly implemented using a single, programmable component, and the length of the circuit in gates is a logarithm of the number of components, its delays are not very dependent on the number of PEs or of components, which is usually between eight and 16.

The And block calculates a logical And between ends of subexpression values, which enables an evaluation of logical formulas in Or priority. The Or block executes a logical Or between the same values, which is used for the evaluation of logical formulas in And priority. As this representation normally represents tuples as a logical formula, the Or block can also check if one tuple at least satisfies the comparison condition with the key tuple. The counting block counts the number of true subexpressions, which is used for the internal sort operation.

Processing elements are all at the

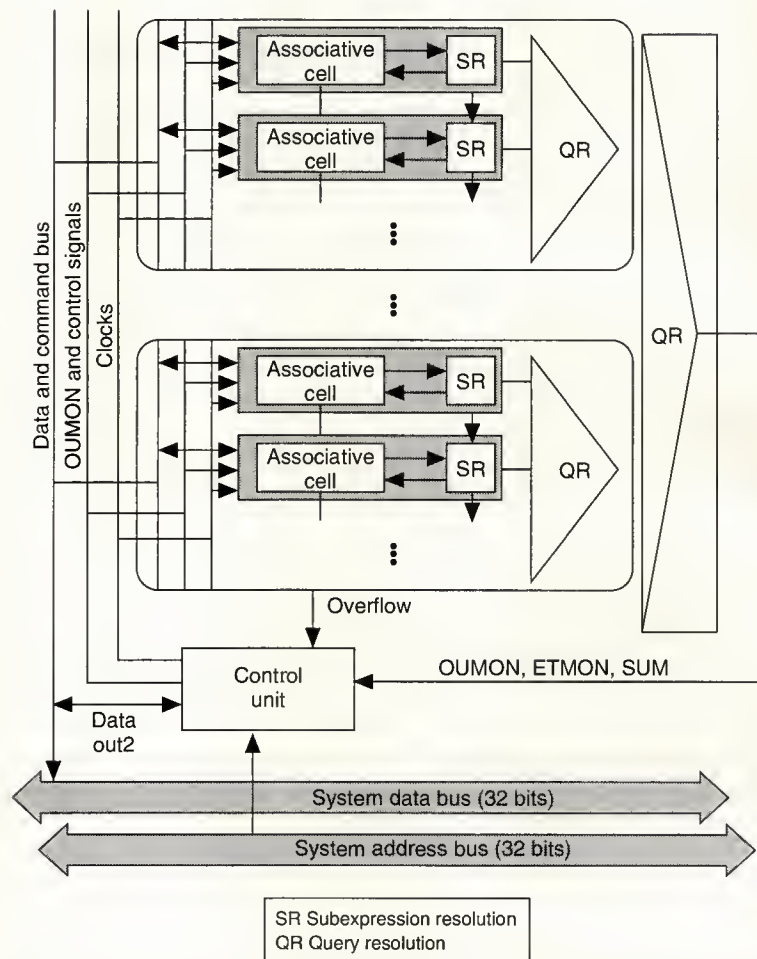


Figure 4. RAPID board.

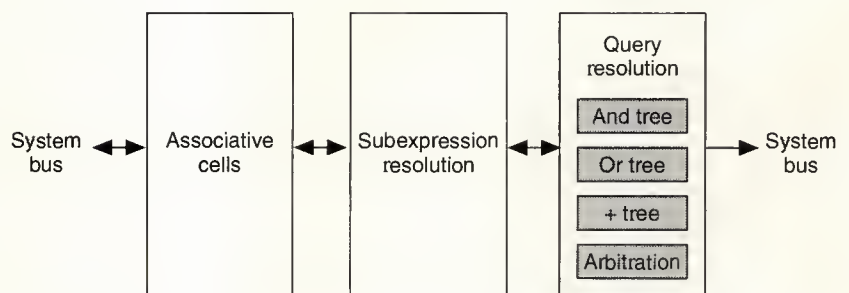


Figure 5. Block diagram of the accelerator.

same address for writes and reads. Writes and reads are done according to a protocol. Data control the circuit. Management of the communication protocol with the host machine bus is at the board level, to make the circuit independent from a given host machine. The board usually produces a clock cycle only if the host processor has written or read data in the circuit at the end of the previous cycle. However, the circuit is synchronized with the host machine.

The associative circuit may be implemented into one or several main components, each including several PEs and part of the other functional blocks. With more than a few tens of PEs per machine, the performance of the accelerator becomes independent from its number of PEs, except for very large relations (more than one million tuples). The subexpression resolution block is fully distributed on the main components, based on one resolution module per PE. Each resolution module cascades with both the previous and next ones. The part of the query resolution block common to several main components is presently constructed with off-the-shelf components.

Processing element. The PE is the basic cell of RAPID. It executes the embedded operations and memorizes data. The external view of the PE in Figure 7 displays its interconnections with its environment. These connections include the internal data bus, command bus, control signals, connections with the previous and next PE, and pins signaling the relative position of the PE in the PE array. Table 1 on page 30 lists descriptions of the corresponding signals.

An internal view of the PE (Figure 8 on page 31) reveals an operative part and a control part. The operative part includes a local memory with 18 words for the storage of a 16-word operand and two attribute numbers. Each word is 16 bits wide, and an attribute number also has 16 bits. The operative part also includes a logical comparator, a predicate generator for PRED values, and several registers. The registers include two for the external and internal configuration of the PE (STATUS1, which holds the instruction word and is read/write, and STATUS2, which holds the PE state and is read-only), two for the interface with the PE environment (INOUT and MCR), and a fifth for the storage of the tuple identifier (IDENT).

The control part is the PE sequencer, which is divided into

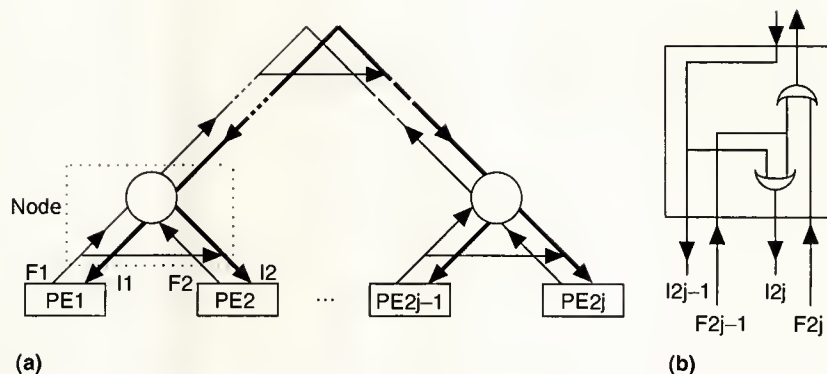


Figure 6. Arbitration block, from Anderson³² (a); close-up of node (b).

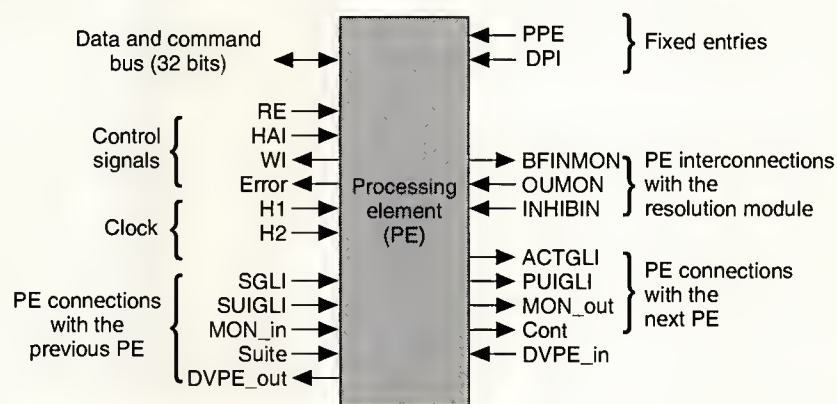


Figure 7. External connections of the PE.

three automata. The test automaton (AUTOT) enables a very fast access to the operand, configuration, and internal registers during normal operation. The primary automaton (AUTOP) schedules the major part of the PE's operations. The auxiliary automaton (AUTOA) helps the main automaton during the loading phase.

Subexpressions resolution. The subexpressions resolution (Figure 9 on page 31) implements a chained And, a chained Or, and a chained Then. MON_in and MON_out are the subexpression values, CONX the connector value, PRED the predicate value, and FINMON the result. We have detailed the subexpression in earlier papers.^{25,26} The response time of the subexpressions resolution circuit equals the propagation time in the logical groups corresponding to the maximum size of a subexpression. We can determine the response time by considering the propagation time on all modules, or by choosing a maximum size for a monomial or subexpression.

An important problem in the circuit design is the processing of large predicates, in which the constant overflows the

Table 1. External PE signals.

Command signals
C1: starting a loading phase at next cycle C2: starting an evaluation phase at next cycle C3: broadcast of a tuple to be evaluated at the next cycle FA: end of an attribute FR: end of the relation FTUP: end of a tuple FMOT: end of a word in a text ADS: presence of a PE address on the data bus CTRL0/T1: opening parenthesis in the normal mode//a bit of the test instruction code in the test mode CTRL1/T2: closing parenthesis in the normal mode//a bit of the test instruction code in the test mode
PE's chain signals
MON_in/MON_out: partial subexpression value propagation Suite/Cont: control transmitted from one PE to another to make it the current PE SGLI/ACTIGLI: activation of the PEs that contains the continuation of a nonanchored search by one PE SUIGLI/PUIGLI: activation of the PE group containing the continuation of a nonanchored search by a group of PEs DVPE_in/DVPE_out: propagation of a tuple overflow signal in a loading phase
Other signals
OUMON: input pin, receives the result of the Or resolution tree RE: PE initialization input HAI: PE entry on the PAUSE mode (especially on the test mode) WI: output pin, bus capture by the PE Error: output pin, error detection by the PE PPE: fixed pin, used to distinguish the first PE of the PEs vector DPE: fixed pin, used to distinguish the last PE the PEs vector

local memory of one PE. Several solutions have been proposed for that problem, including an And of comparisons on several operand parts identified by an index,²² the addressing of a group of cells corresponding to an address where low-weight bits are masked,²³ or the transfer of a control token from one byte to the next in ISSP.^{20,21} We pass control from

one PE to the next when the comparison on the current operand part ends, and when the comparison result is equality. The corresponding predicate value is then forced to the neutral value for the subexpression resolution, true-in-And priority. The only significant value of PRED is that of the first PE that returns an inequality comparison or the value of PRED in the last PE. The PRED values for the next PEs in the same predicate—if any exist—remain at the neutral value.²⁵

Rapid versions. We designed version V0 to prove the circuit feasibility. It implements all relational operations extended to sorting. It was customized using a Silvar Lisco CAD software. A component contains a single PE, in a 68-pin CLCC package. Fifteen of the pins are free for further versions. Table 2 gives the characteristics of this version.

With a host machine powerful enough to saturate the accelerator, V0 executes a $1,000 \times 1,000$ -tuples JOIN in 4 ms. However, the limitation of the total number of PEs on an actual board implies an important overhead in the case of large relations.

Implementation of V0 effectively began in 1988, but stopped in 1989 due to the departure of the logic designer. At this time, we had only a partial simulation. Checks at logic and layout levels were resumed in spring 1990 by a PhD student. The ES2 Company fabricated it in October 1990.

We designed most of V1 from scratch. V1.1 is compatible with the SQL-1 standard and implements four PEs on each component. A practical board with eight main components may then include 32 PEs, enough for current performance standards. We designed this version with standard cells and a 1.5-micron technology, using the Cadence CAD package.

V1.1 includes text retrieval, detection, and processing of null values according to the SQL-1 standard and the processing of overflows. Modifications of the PROJECTION operation enable it to use the AGGREGATE with GROUPED BY function, according to the method described earlier. We also improved testability. We added scan paths, which make it possible to separately scan four parts of the circuit. Using the test automaton, we can rapidly access the internal states of the circuit during normal operation and therefore run fast test programs. The effective version design started in summer 1990 with one PhD student, and we expect the circuit to be fabricated by the end of 1991.

Environment and performance

With the RAPID accelerator, access to data, hashing, data transfer to the accelerator, and results transfer into result relations, are done by a software kernel executed by the host processor. This kernel is composed of a multiuser transactions manager that manages threads (lightweight processes), a query machine that executes query operations, and an object manager that does object creation and access. The thread management by the software kernel guarantees a low-cost, atomic execution of operations by the accelerator, that

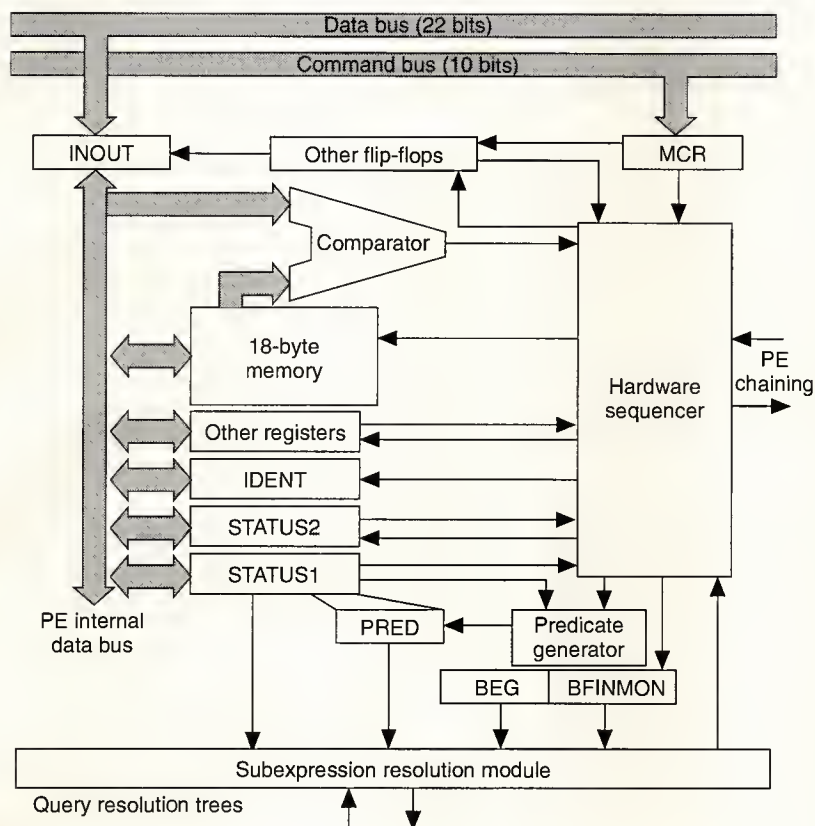


Figure 8. Internal view of the processing element.

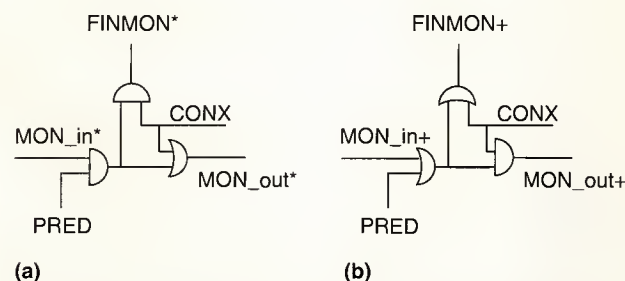


Figure 9. Subexpressions resolution: And priority (a); Or priority (b).

is to say, an execution in which a suboperation is never pre-empted. This atomicity is needed for performance reasons. The accelerator executes the operations. However, in the case of selections, preselections of the most selective attribute are done using an index. Software kernel performance is a

major factor of RAPID's efficiency.

We developed the first version of the software kernel on a Sun 3/50, with a recent object manager. We measured the duration of the 1,000 × 1,000-JOIN (a JOIN of 1,000 tuples per 1,000 tuples returning 1,000 result tuples) in the software configuration that corresponds to the use of the accelerator. We have not used the accelerator board in the Sun workstation. (One version of the board has been implemented in a Macintosh II.) The software operation time is 1.23 seconds. The product of this time and the number of MIPS of the machine (about 2.5) is better than the throughput of the Gamma machine, which is a well-known parallel machine that uses about 7 MIPS to execute this operation in memory.³³ The time lapse in the accelerator is very small, about 4 ms with the first version of the circuit.

We are improving the performance with a novel, performance-oriented, object manager. We have completed the design of this object manager and estimate it will reduce the access time by approximately one order of magnitude in an average case. This object manager is a critical module. By using it and some complementary software improvements, the software duration of the reference operation should drop to about 200 ms, while the time spent in the hardware should be about 2.58 ms with the V1 version.

Figure 10 on page 32 shows the improvement ratio at a given number of MIPS, based

Table 2. Features of RAPID V0.

Configuration	Number of PEs × 16 words × 22 bits
Instruction set	7 instructions
Cycle time	120 ns
Supply voltage	5 V
VLSI process technology	2-μm CMOS with double aluminium layers
Number of devices	11,000
Chip size	5.3mm × 5.2mm
Number of pins	50
Package	68-pin CLCC

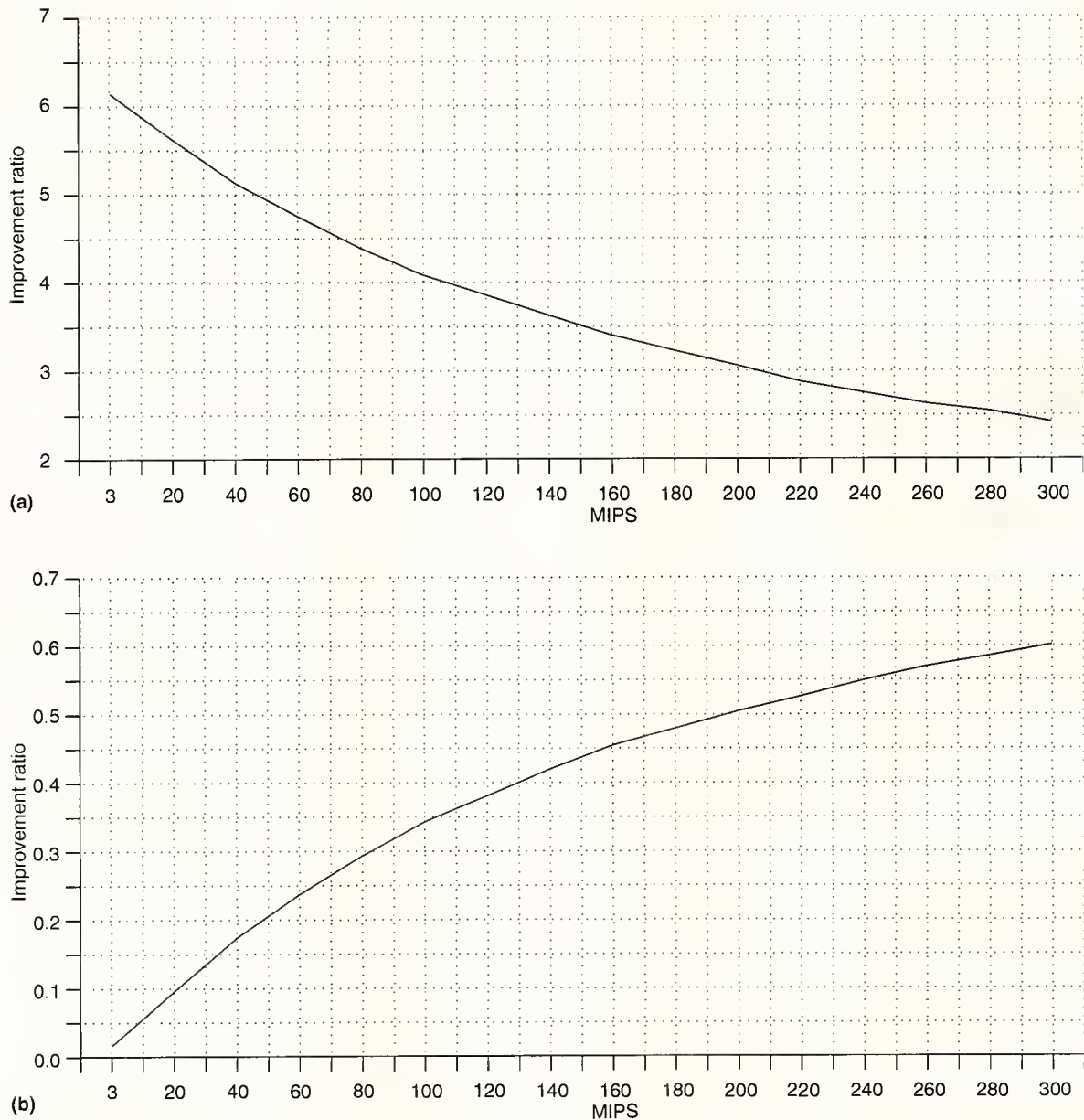


Figure 10. Improvement ratio vs. host processing power (a); accelerator utilization ratio vs. host processing power (b).

on our assumptions. We assume the host machine throughput in Mbytes equals the number of MIPS. But the throughput needed for the circuit I/Os is much smaller, about 10^5 bytes/s \times MIPS. Clearly, the accelerator can significantly improve performance even on a 200-MIPS host machine.

THE RAPID-1 IS AN ASSOCIATIVE ACCELERATOR for database operations. Its instruction set includes relational operations extended to SORT and AGGREGATE with GROUPED BY. Data do not remain in the associative circuit, except for very short durations. Programming is a mere trans-

lation of query predicates, though an adequate software kernel is needed for efficiency.

The circuit uses a small number of associative cells with hardwired control. It speeds up database systems by a factor of five, versus most referenced database systems. The acceleration ratio will still be very significant with the emerging class of new microprocessors, which may have a peak power about or above 100 MIPS. The accelerator will also speed up parallel database systems using these processors.

These results show that the choice of an associative circuit with hardwired control, that is controlled by data, with bit-parallel comparisons for all operations, is critical for the use of the accelerator with new generations of processors. Other solutions are possible but would lead to machines with weaker performance (due to multiple-instruction transfers) and less homogeneity when inequality predicates are used (such as for sorting).

We hope to see this accelerator evolve further to include the processing of binary strings, fault tolerance, a feasible performance increase by a factor of 10 to meet the needs of the next generation of microprocessors, and the design of components with more parallelism. ■

Acknowledgments

We thank all the members of the RAPID team for their very fruitful contribution and especially E. Abecassis, D. Donsez, G. Fouquet, D. Guidot, H. He, and J. Penne. We also thank the anonymous referees for fruitful comments resulting in several improvements of this article.

The MASI Laboratory is an associated unit of the Centre National de la Recherche Scientifique. The RAPID project has been partly financed by Agence Nationale de Valorisation de la Recherche (ANVAR), and is now partly financed by the French Ministère de la Recherche et de la Technologie (AASI-89 program), and by the PRC-AMN, a French Cooperative Research Program in Novel Machine Architectures at the Centre National de la Recherche Scientifique.

References

1. H. Boral and D.J. Dewitt, "Database Machines: An Idea Whose Time Has Passed?" *Proc. Third Int'l Workshop Database Machines*, Springer-Verlag, New York, 1983, pp. 167-187.
2. Laguna Beach Group, "The Laguna Beach Report: Future Directions in DBMS Research," *SIGMOD Record*, Vol. 18, No. 1, Mar. 1989.
3. G. Musgrave, ed., *Proc. VLSI*, International Federation for Information Processing, Vol. 10.S, Munich, Aug. 1989.
4. *DBC/1012 Data Base Computer Concept and Facilities*, Document No. C02-000100, Teradata Corp., 1983.
5. Tandem Performance Group, "A Benchmark of Non-Stop SQL on Debit-Credit Transaction," *Proc. ACM-SIGMOD*, ACM, N.Y., 1988, pp. 337-341.
6. M. Kitsuregawa, "Implementation of LSI Sort Chip for Bimodal Sort Memory," *Proc. VLSI*, International Federation for Information Processing, 10.S, 1989, pp. 285-294.
7. M. Abdelguerfi and A.K. Sood, "A Bus-Connected Cellular Array Processing Unit for Relational Database Machines," *Proc. Fifth Int'l Workshop Database Machines*, Kluwer Academic Publishers, Boston, 1987, pp. 188-201.
8. K.C. Lee and G. Herman, "A High-Performance VLSI Data Filter," *Proc. Fifth Int'l Workshop Database Machines*, Kluwer Academic Publishers, 1987, pp. 251-268.
9. D.J. DeWitt and H. Boral, eds., "Database Machines," *Proc. Fourth Int'l Workshop Database Machines*, Springer-Verlag, 1985.
10. M. Kitsuregawa, ed., *Proc. Fifth Int'l Workshop Database Machines*, Kluwer Academic Publishers, 1987.
11. H. Boral and P. Faudemay, "Database Machines," *Proc. Sixth Int'l Workshop Database Machines*, Lecture Notes in Computer Science, No. 368, Springer-Verlag, 1989.
12. J. Minker, "An Overview of Associative and Content-Addressable Memory Systems and a KWIC Index to the Literature," *Computing Reviews*, Oct. 1971.
13. K.J. Thurber and L.D. Wald, "Associative and Parallel Processors," *Computing Surveys*, Vol. 7, No. 4, Dec. 1975, pp. 215-256.
14. L. Chisvin and R.J. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer*, Vol. 22, No. 7, July 1989, pp. 51-64.
15. D. Bitton et al., "Parallel Algorithms for the Execution of Relational Parallel Operations," *Trans. Database Systems*, Vol. 8, No. 3, Sept. 1983, pp. 324-353.
16. M. Kitsuregawa et al., "Application of Hash to Database Machine and its Architecture," *New Generation Computing*, Vol. 1, No. 1, 1983, pp. 63-74.
17. S.H. Lavington and R.A.J. Davies, "Active Memory for Managing Persistent Objects," *Proc. Int'l Workshop on Computer Architectures to Support Security and Persistence*, Springer-Verlag, May 1990, pp. 137-154.
18. S.H. Lavington and J. Robinson, "Exploiting Data Parallelism in Knowledge-Based Systems," Research Report CSM-158, Department of Computer Science, Essex University, U.K., 1990.
19. M. Abdelguerfi, "Special Function Unit for Statistical Aggregation Functions," *Proc. Sixth Int'l Workshop on Database Machines*, Lecture Notes in Computer Science, No. 368, Springer-Verlag, 1989, pp. 187-201.
20. K. Takahashi, H. Yamada, and M. Hirata, "A String Search Processor LSI," *J. Information Processing*, Vol. 13, No. 2, 1990, pp. 183-189.
21. K. Takahashi et al., "A New String Search Hardware Architecture for VLSI," *Proc. 13th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., June 1986, pp. 20-27.
22. T. Ogura et al., "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, Vol. 24, No.

- 4, Aug. 1989, pp. 1014-1027.
23. J.P. Wade and C.G. Sodini, "A Ternary Content Addressable Search Engine," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1003-1013.
24. S.H. Lavington et al., "Hardware Memory Management for Large Knowledge Bases," *Proc. PARLE (Parallel Architectures and Languages Europe)*, Vol. I, Lecture Notes on Computer Science, No. 258, Springer-Verlag, 1987.
25. P. Faudemay et al., "The Database Processor RAPID," *Proc. Fifth Int'l Workshop on Database Machines*, Kluwer Academic Publishers, 1987, pp. 171-187.
26. P. Faudemay, *Un Processeur VLSI pour les Operations de Bases de Donnees* (A VLSI Processor for Database Operations), PhD thesis, University Pierre et Marie Curie, Paris, 1986.
27. G. Salton, E. Fox and H. Wu, "Extended Boolean Information Retrieval," *Comm. ACM*, Vol. 26, No. 12, Dec. 1983, 1022-1036.
28. C. Stanfill and B. Kahle, "Parallel Free-Text Search on the Connection Machine," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986, pp. 1213-1228.
29. H. He, P. Faudemay, and L. Chen, "Le tri, la projection et les agregats dans la machine bases de donnees RAPID," ("Sorting, Projection, and Aggregates in the RAPID Database Machine."), research report, Institut Blaise Pascal, 1990.
30. K. Iverson, *A Programming Language*, John Wiley & Sons, N.Y., 1962.
31. M. Penazola and E. Ozkaran, "Integrating Integrity Constraints With Database Filters Implemented in Hardware," *Proc. Sixth Int'l Workshop on Database Machines*, Lecture Notes in Computer Science, No. 368, Springer-Verlag, 1989, pp. 230-250.
32. G.A. Anderson, "Multiple Match Resolvers: A New Design Method," *IEEE Trans. Computers*, Vol C-23, No. 12, Dec. 1974, pp. 1317-1322.
33. D.A. Schneider and D.J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," *Proc. ACM-SIGMOD*, ACM, 1989, pp. 110-121.
34. M. Motomura et al., "A 1.2-Million-Transistor, 33-MHz, 20-b Dictionary Search Processor (DISP) ULSI with a 160-Kbyte CAM," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, Oct. 1990, p. 1158-1165.



Pascal Faudemay is project manager of the RAPID project at MASI Laboratory and an engineer at the Centre National de la Recherche Scientifique. He designs performance-oriented software and hardware architectures for databases. His interests

include microarchitecture, databases, object servers, real-time processing, and parallel and distributed systems. Faudemay earned BS and MS degrees from University Paris I and a PhD in computer science from University Pierre et Marie Curie in 1986. He is a member of the IEEE Computer Society and ACM.



Mongia Mhiri is a PhD student in computer science at University Pierre et Marie Curie. She works on version V1 of the RAPID associative circuit at MASI Laboratory. Mhiri received BS and MS degrees in computer science from Grenoble University in 1989.

Address questions concerning this article to Pascal Faudemay, Laboratoire MASI, University Pierre et Marie Curie, 4 Place Jussieu, 75252 Paris cedex 05, France; or via e-mail at faudemay@masi.ibp.fr.

Expedited delivery

is available to all members residing outside the USA, Canada, and Mexico. We invite you to take advantage of this service providing delivery of your magazine weeks earlier.

For information on this service and its cost, contact:

Expedited Delivery
IEEE Computer Society
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264 USA
Phone: (714) 821-8380
FAX: (714) 821-4010

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156

Medium 157

High 158



A Fine-Grain Architecture for Relational Database Aggregation Operations

This design and simulation of a bit-sliced processor for relational database aggregation functions addresses an important, computationally expensive problem in database computers. The slice processor takes two tuples as inputs (one bit at a time) and returns two bits as outputs every clock cycle. A larger aggregation unit uses a number of identical slice processors, connected according to odd-even network topology, to achieve improved performance on a parallel pipelined processor.

M. Abdelguerfi

University of New Orleans

A.K. Sood

George Mason University

Since its introduction by Codd in 1970,¹ the relational database model has achieved wide acceptance. While allowing a high degree of data independence, the model provides a theoretical means to deal with consistency and redundancy problems. This is possible mainly because data is represented in two-dimensional tables that can be designed using the normalization theory.

In the relational database model, data can be manipulated through a small but powerful set of operators. These operators are the relational algebra operations (such as PROJECTION, JOIN, and SET) and aggregation operations. Most of these operations are computationally intensive and algorithm design for efficient implementation plays an important role in the design of database computers. Efficient processing has a determining effect on system performance. As a result, the operations have been the subject of intense study in the development of relational database management systems.

For instance, to maximize the performance of the natural JOIN operation, several algorithms have been developed.² While these methods tended to be effective in conventional von

Neumann computer systems, a closer look reveals great opportunity for concurrent processing. With appropriate hardware and system organization, such concepts as parallelism and pipelining can significantly enhance the performance of these operations.

We designed and simulated a special-purpose unit for such aggregation functions as Sum, Count, and Average using a one-bit slice processor as a basic component. (We do not consider the remaining two statistical aggregation functions Min and Max here as they are not computationally intensive.) A larger unit can be easily designed by connecting several identical slice processors according to odd-even network topology.³ In this unit the tuples are input, processed, and output in parallel, one tuple at a time. The processing elements operate on tuples bit by bit. As a result, we refer to this system as parallel bit-level pipelined architecture.⁴

The main advantage of this approach is that the memory requirement of each slice processor is very small and is independent of input size. Since input operands process one bit at a time, we reduce the amount of hardware in each slice processor. As a result, a large number of slice

processors can be integrated on one VLSI chip. The proposed design exhibits several attractive features.

- **A simple, basic slice processor.** The system makes use of only one type of simple slice processor. Each slice operates on data bit by bit, simplifying design and verification of the circuit.
- **Overlap of data I/O and processing.** Data processing time is completely overlapped with data input and output to and from each slice.
- **Static interconnection network.** For ease of implementation, the design uses a static interconnection between the different slices rather than a dynamic network. This connection allows the system to process several data streams in a pipelined manner.
- **High throughput.** Several bit-serial slices operating in parallel achieve high throughput.
- **Low pin count.** The input and output of operands one bit at a time, to and from the slices, ensure a low count.
- **Design is independent of tuple size.**
- **Limited interconnection requirement.** Bit-serial computation limits interconnection needs.

Output of the tuples is completely overlapped with their input and processing times. Therefore, a tuple stream input to the unit processes in a pipelined way, one-bit per tuple at a time. Several different input streams of tuples can also be processed in a pipelined manner. That is, the processing of a new input stream of tuples can be initiated while the previous stream is still being processed. Consequently, the proposed unit achieves pipelining at both the tuple and input stream levels.

We use a SIMD (single-instruction stream, multiple-data) architecture to perform statistical aggregation functions. In contrast to the parallel pipelined query processor approach in Kim et al.,⁵ sorting, aggregation, and duplicate marking are performed concurrently. One network topology and one type of processing element (slice) implement these functions. The design in Kim et al. uses three different network topologies and processes parallel tuples one bit at a time.

The processor

Let's examine the design of our Sum processing unit. The same unit will be used to perform the COUNT and AVERAGE operations.

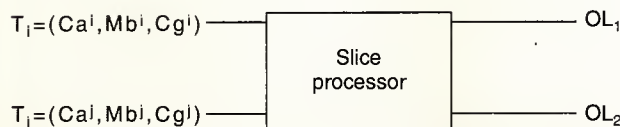


Figure 1. A slice processor.

```

Procedure SUM ( (Cai, Mbi, Cgi), (Caj, Mbj, Cgj) )
/* Two reduced tuples Ti = (Cai, Mbi, Cgi) and Tj = (Caj, Mbj, Cgj)
are input to a slice */
begin
  if Cgi > Cgj then
    begin OL1 := (Cai, Mbi, Cgi) ; OL2 := (Caj, Mbj, Cgj) ;
  elseif Cgi < Cgj then
    begin OL1 := (Caj, Mbj, Cgj) ; OL2 := (Cai, Mbi, Cgi) ;
  else
    if Mbi AND Mbj = 1 then
      OL1 := (Cai + Caj, Mbi, Cgi) ; OL2 := (-, 0, Cgi) ;
    elseif Mbi AND Mbj = 1 then
      OL1 := (Caj, Mbj, Cgj) ; OL2 := (-, Mbj, Cgj) ;
    else
      OL1 := (Cai, Mbi, Cgi) ; OL2 := (Caj, Mbj, Cgj) ;
    endif ;
  endif ;
end;

```

Figure 2. Procedure Sum algorithm.

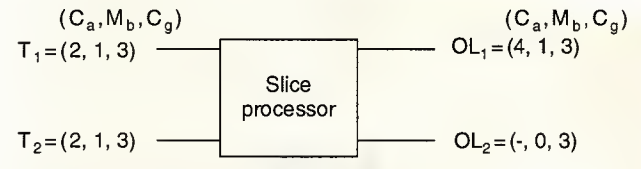


Figure 3. Example of a Sum algorithm.

Methodology. To each reduced tuple, we associate a mark bit we call Mb. Each one-bit slice processor takes as inputs two reduced tuples (see Figure 1). The slice processor compares the two Group-By values (Cgⁱ, Cg^j). It manipulates the aggregation values (Caⁱ, Ca^j) and mark bits (Mbⁱ, Mb^j) according to the result of the comparison phase. The resulting reduced tuples are output through OL₁ and OL₂. The mark bits identify the qualified tuples, which are output with a mark bit set equal to 1. Thus, the processor filters out duplicate tuples by examining their mark bit. Figure 2 shows the algorithm performed by the slice processor.

An example appears in Figure 3 in which two tuples (T₁ and T₂) are input to the slice processor. These tuples are both qualified (their mark bits are both 1) and have the same Group-By value. Therefore, their aggregation columns will be summed, and one of the tuples will be disqualified by resetting its mark bit to 0.

Hardware implementation. Figure 4 depicts a block diagram of a one-bit slice processor that contains five flags: F₁, F₂, F₃, F₄, F₅. The two reduced tuples are input, processed, and output one bit at a time. Processing starts with the Group-By values Cg^m = {Cgk^m, Cg2^m, ..., Cg1^m} m = i, j, which are

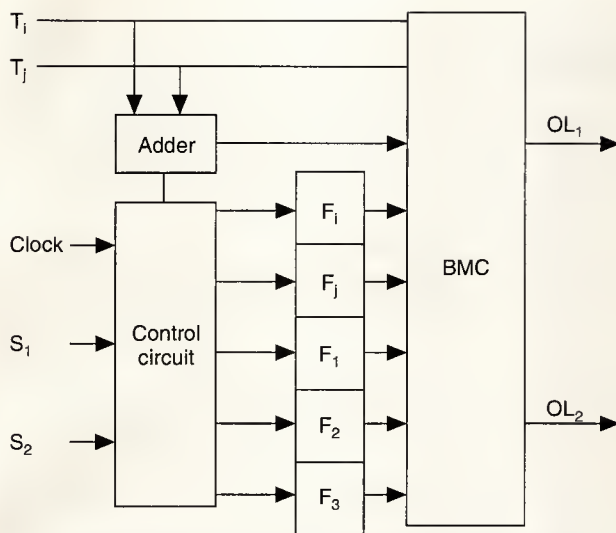


Figure 4. Block diagram of a slice processor.

input to the slice processor bit by bit starting from the most significant bits, or MSBs, (Cgk^i , Cgk^j). (Here, k represents the number of bits in the binary representation of the Group-By values.) This makes the bit-serial comparison of the two Group-By values easier to perform.

Next, mark bits Mb^i and Mb^j are input, followed by the aggregate values $Ca^m = \{Ca1^m, Ca2^m, \dots, Cap^m\}$, $m = i, j$. Aggregate values must be added serially, bit by bit, starting from the least significant bits, or LSBs. Therefore, unlike the Group-By values, the aggregate values are input starting from the LSBs ($Ca1^i$, $Ca1^j$). The number of bits in the binary representation of the aggregate values is p . Notice that the Group-By values comparison, the manipulation of mark bits, and the summation of aggregate values completely overlap with the input and output of the reduced tuples to and from the slice processor.

Control and synchronization are achieved through two control signals S_1 and S_2 . S_1 , a start signal, is applied to the slice processor at time instant t_0 for one clock cycle. During this time instant, the slice processor resets its flags. At the next time instant, the slice processor begins the processing of the reduced tuples present at its two inputs.

Control signal S_2 indicates the completion of Group-By values comparison. This signal is applied to the slice processor at time instant t_k for one clock cycle. At the same time instant, the two mark bits are input to the slice processor. From time instant t_{k+1} to t_{k+p} , the aggregate values are input to the slice processor.

The synchronous serial adder computes the sum ($Sum = Ca^i + Ca^j$). CF denotes the adder carry flag (not shown in the block diagram). Flags F_i and F_j store mark bits Mb^i and Mb^j . Control signal S_2 sets flag F_3 , which indicates that Group-By values

comparison has been completed. Finally, flags F_1 and F_2 store the outcome of the comparison according to Table 1.

The Bit Manipulation Circuit (BMC) determines the two outputs $OL_m = (Cgk^m, Cg2^m, \dots, Cg1^m, Mb^m, Ca1^m, Ca2^m, \dots, Cap^m)$ and $m = 1, 2$. The Bit Serial Sum algorithm (see the box) computes each output bit. The slice processor starts the output process one clock cycle after the input process begins. Thus, the output process completely overlaps both the input and processing times. Note that the hardware algorithm is independent of the tuple size. Because tuple size varies from one application to another, this characteristic is crucial. As shown in Figure 5, by controlling the time interval between control signals S_1 and S_2 , the slice processor can be tuned to a particular tuple size.

We designed a slice processor using 10 D flip-flops and about 72 gates. Figures 6 and 7 show an example (with $p = 2$ and $k = 4$) and its simulation. In this example, the simulation starts at the trailing edge of the first appearance of S_1 . The output process starts one cycle after the input process begins.

Implementing other aggregation operations. The Count and Average functions can be implemented using the Sum unit. For the COUNT operation, each reduced tuple will be input to the Sum unit with a Ca value set equal to 1. Query

Table 1. Flags F_1 and F_2 .		
F_1	F_2	Condition
0	0	$Cg^i = Cg^j$
0	1	—
1	0	$Cg^i > Cg^j$
1	1	$Cg^i < Cg^j$

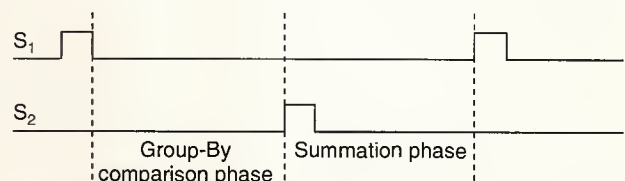


Figure 5. Timing diagram.

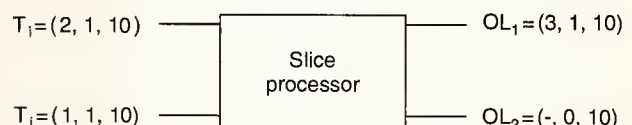


Figure 6. Example of a one-slice processor operation.

The Bit-Serial Sum Algorithm

Procedure Bit-Serial SUM (T_i, T_j)

/* Two tuples $T_i = (Cgk^i, \dots, Cg2^i Cg1^i Mb^i Ca1^i Ca2^i, \dots, Cap^i)$ and $T_j = (Cgk^j, \dots, Cg2^j Cg1^j Mb^j Ca1^j Ca2^j, \dots, Cap^j)$ are processed one bit at a time by the slice processor. The outputting of the result is completely overlapped with the inputting process. The slice processor two outputs are $OL_m = (Cgk^m, \dots, Cg2^m Cg1^m Mb^m Ca1^m Ca2^m, \dots, Cap^m)$, $m=1,2$ */
if $S_1 = \text{true}$ then begin /* Initialize flags (the duration of S_1 is one clock cycle)*/

$F1 := 0$; $F2 := 0$; $F3 := 0$; $Fi := 0$; $Fj := 0$; $n := k$;

while $S_2 = \text{false}$ and $n \geq 1$ loop /* Start the GROUP BY comparison phase */

if $(C_{gn}^i > C_{gn}^j)$ then

if $(F1 = 0)$ then $C_{gn}^1 := C_{gn}^i$; $C_{gn}^2 := C_{gn}^j$; $F1 := 1$;

elseif $(F2 = 0)$ then $C_{gn}^1 = C_{gn}^i$; $C_{gn}^2 := C_{gn}^j$;

else $C_{gn}^1 := C_{gn}^j$; $C_{gn}^2 := C_{gn}^i$;

endif;

elseif $(C_{gn}^i < C_{gn}^j)$ then

if $(F1 = 0)$ then $C_{gn}^1 := C_{gn}^j$; $C_{gn}^2 := C_{gn}^i$; $F1 := 1$;

$F2 := 1$;

elseif $(F2 = 0)$ then $C_{gn}^1 := C_{gn}^j$; $C_{gn}^2 := C_{gn}^i$;

else $C_{gn}^1 := C_{gn}^i$; $C_{gn}^2 := C_{gn}^j$;

endif;

else $C_{gn}^1 := C_{gn}^i$; $C_{gn}^2 := C_{gn}^j$;

endif;

$n := n - 1$;

end;

/* S_2 is now high for one clock cycle. During this Phase (called mark bit manipulation phase) the two mark bits are stored (in Fi and Fj) and a mark bit manipulation is performed. The phase duration is one clock cycle*/

$F_i := M_b^i$; $F_j := M_b^j$; $CF := 0$;

if $(F1 = 0)$ then $M_b^1 := (M_b^i \text{ or } M_b^j)$; $M_b^2 := 0$;

else $M_b^1 := M_b^i$; $M_b^2 := M_b^j$;

endif;

$m := 1$

/* Now the summation phase begins. This phase will last until a new start signal (S_1) is applied to the processor*/

while $S_1 = \text{false}$ and $m \leq p$ loop

if $(F1 = 0)$ and $(Fi = 1)$ and $Fj = 1$ then /* "-" refers to don't care*/

$Cam^1 := Cam^i \text{ XOR } Cam^j \text{ XOR } CF$; $Cam^2 := -$; CF

$:= Cam^i \text{ and } Cam^j$;

elseif $((F1 = 0) \text{ and } (Fi = 0) \text{ and } (Fj = 1))$ or $(F2 = 1)$ then

$Cam^1 := Cam^i$; $Cam^2 := Cam^j$;

else $Cam^1 := Cam^j$; $Cam^2 := Cam^i$; endif;

$m := m + 1$;

end;

endif;

end Bit-Serial SUM;

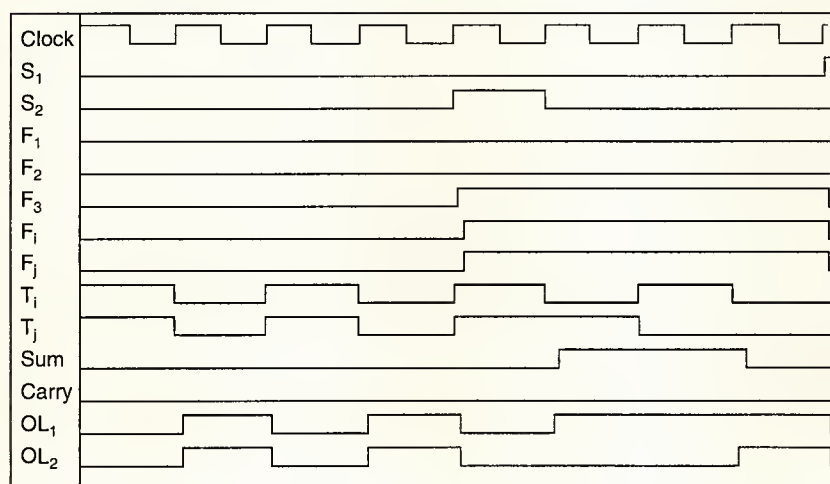


Figure 7. Simulation of the example in Figure 6.

3 in the Aggregation Functions box illustrates the AVERAGE operation, which is a combination of both Sum and Count. The AVERAGE operation proceeds as follows. To each reduced tuple (Ca, Mb, Cg) , we attach a new attribute value Cc . This attribute value is set initially to 1. The augmented tuple (Cc, Ca, Mb, Cg) is input one bit at a time to the Sum unit. The Cc value is input immediately following input of the Ca values. The first $(k+1)$ bits output from the Sum slice processor represent a Group-By value followed by a mark bit. The next p bits output from the unit represent a Ca value. Finally, the remaining output bits represent a Cc value.

Designing a Sum processor

A one-bit slice processor can process two tuples one bit per tuple at a time.

Aggregation functions

In database management applications, we often want to categorize the tuples of a relation by the values of a set of attributes and extract an aggregated characteristic of each category. We call these database management tasks aggregation functions. For instance, the SQL data language includes the following built-in aggregation functions: Sum, Count, Average, Min, Max. The attributes used for the categorization are referred to as Group By columns.

Consider the relation *professor* as *professor (faculty, department, salary)*. Each tuple of this relation gives the name of a faculty person, the department, and the academic year salary. Table A lists an instance of the relation *professor*.

In this relation a row such as <Smith, Electrical Eng., \$39,000> is referred to as a tuple. There are no duplicate tuples. Consider the following queries:

Query 1: How many faculty are in each department? The result is to be ordered by *department*.

This query requests a count of the number of faculty for each department. *Faculty* are therefore categorized according to the attribute *department*. As a result, *department* is referred to as a Group-By attribute. In SQL, the above query is formulated as follows:

```
Q1: Select department, Count (faculty)
    From professor
    Group By department
    Order By department
```

The result of applying the Count aggregation function is a new relation with two attribute names. They are a Group-By attribute (*department*, in this case) and a new attribute called Count. The tuples are ordered lexicographically in ascending order according to the Order-By attribute (*department*). The resulting relation, called *Man-power*, is

<u>department</u>	<u>Count (faculty)</u>
Computer Sc.	4
Electrical Eng.	3
Mechanical Eng.	2

Query 2: What is the total salary for all faculty in each department? The result is to be ordered by *department*.

The SQL expression for this query is

```
Q2: Select department, Sum (salary)
    From professor
    Group By department
    Order By department
```

Table A. Instance of the relation *professor* (*faculty, department, salary*).

<i>faculty</i>	<i>department</i>	<i>salary</i>
Smith	Electrical Eng.	\$39,000
Joe	Mechanical Eng.	\$35,000
Susan	Computer Sc.	\$36,000
Erick	Electrical Eng.	\$38,000
Paul	Electrical Eng.	\$37,000
Johannes	Computer Sc.	\$65,000
Rick	Computer Sc.	\$32,000
Gerard	Computer Sc.	\$43,000
Kenneth	Mechanical Eng.	\$40,000

The result of the Sum aggregation function is a new relation called *payroll*.

<u>department</u>	<u>Sum(salary)</u>
Computer Sc.	\$176,000
Electrical Eng.	\$114,000
Mechanical Eng.	\$75,000

Query 3: What is the average salary in each department? The result is to be ordered by *department*.

The SQL formulation of query 3 is

```
Q3: Select department, Average (salary)
    From professor
    Group By department
    Order By department
```

Applying the Average operator to the relation *professor* yields the relation *median-salary*.

<u>department</u>	<u>Average (salary)</u>
Computer Sc.	\$44,000
Electrical Eng.	\$38,000
Mechanical Eng.	\$37,500

Note that the AVERAGE aggregation operation combines both the SUM and COUNT operations. Another way of representing the relation median is as follows:

<u>department</u>	<u>Sum (salary)</u>	<u>Count(faculty)</u>
Computer Sc.	\$176,000	4
Electrical Eng.	\$114,000	3
Mechanical Eng.	\$75,000	2

continued on next page

Aggregation functions (continued)

The Average (*salary*) for each department can be easily obtained by performing the following operation:

$$\text{Average (salary)} = \text{Sum (salary)} / \text{Count (faculty)}$$

Notice that the COUNT operation can be performed using the Sum operation by adding an extra attribute (*extra*) to the relation *professor*. For each tuple of the new relation denoted *professor**, the extra attribute value is set to 1. An instance of the new relation appears in Table B.

Query 4. The following Sum SQL statement is equivalent to Count statement Q1:

Q4: Select *department*, Sum (*extra*)
 From *professor**
 Group By *department*
 Order By *department*

These remarks are important from a hardware implementation viewpoint. They imply that a Sum hardware unit, if designed efficiently, can also be used to process the Count and Average aggregation functions.

From these remarks, we can see that the processing of aggregation functions involves in general two columns: Group By (*Cg*) and aggregation (*Ca*). For instance, in Q2, the

Group-By column is *department*, and the aggregation column is *salary*. A tuple consisting only of these two attributes will be referred to as reduced tuple. During the process of performing Sum and Average, a new column called the *sum column* (*Cs*) is generated. Each value in *Cs* gives the sum of all *Ca* values corresponding to a distinct *Cg* value. Also, in the process of performing Count and Average, a new column referred to as *count column* *Cc* is generated. Each *Cc* value gives the number of occurrences of a distinct value in the Group-By column.

Table B. Instance of the relation *professor (*faculty*, *department*, *salary*, *extra*).**

<i>faculty</i>	<i>department</i>	<i>salary</i>	<i>extra</i>
Smith	Electrical Eng.	\$39,000	1
Joe	Mechanical Eng.	\$35,000	1
Susan	Computer Sc.	\$36,000	1
Erick	Electrical Eng.	\$38,000	1
Paul	Electrical Eng.	\$37,000	1
Johannes	Computer Sc.	\$65,000	1
Rick	Computer Sc.	\$32,000	1
Gerard	Computer Sc.	\$43,000	1
Kenneth	Mechanical Eng.	\$40,000	1

Processing a larger number of tuples in one pass requires several slice processors connected according to Batcher's odd-even network topology.³

The odd-even network topology has been used to design several special-purpose units. Batcher used it to design a sorting network, and Sood et al.⁶ used this structure to implement relational algebra operations. Abdelguerfi et al.⁴ uses the same unit for signal processing. Here, we show how a larger Sum unit can be obtained by connecting a number of slice processors using the odd-even topology.

In this type of network topology, an *n*-input Sum unit is composed of

$$\left(\frac{n(\log^2 n - \log n + 4)}{4} - 1 \right) \quad (1)$$

identical slice processors connected according to the odd-even network topology. (Throughout this article we refer to

\log_2 as \log .) The longest path in an *n*-input Sum processing unit is composed of $\lceil \log n (\log n + 1) / 2 \rceil$ levels. You will recall that the basic component of the Sum unit is a pipelined one-bit slice processor. Slices of each level take as inputs two bits (one per reduced tuple) from the preceding level, process the two bits in one clock cycle, and output two new bits to slices

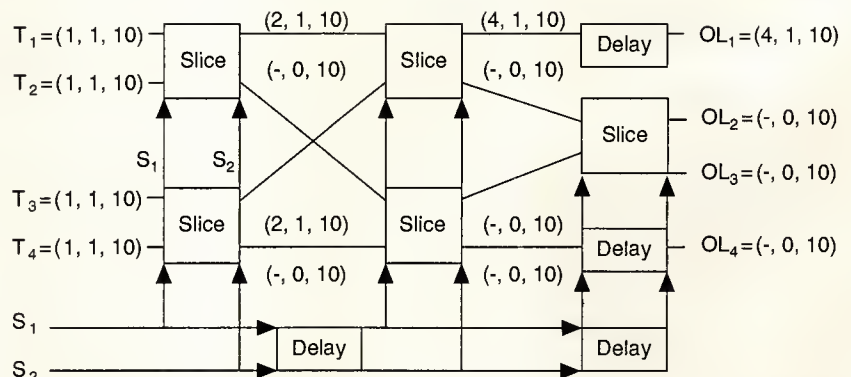


Figure 8. Example with a four-input Sum unit.

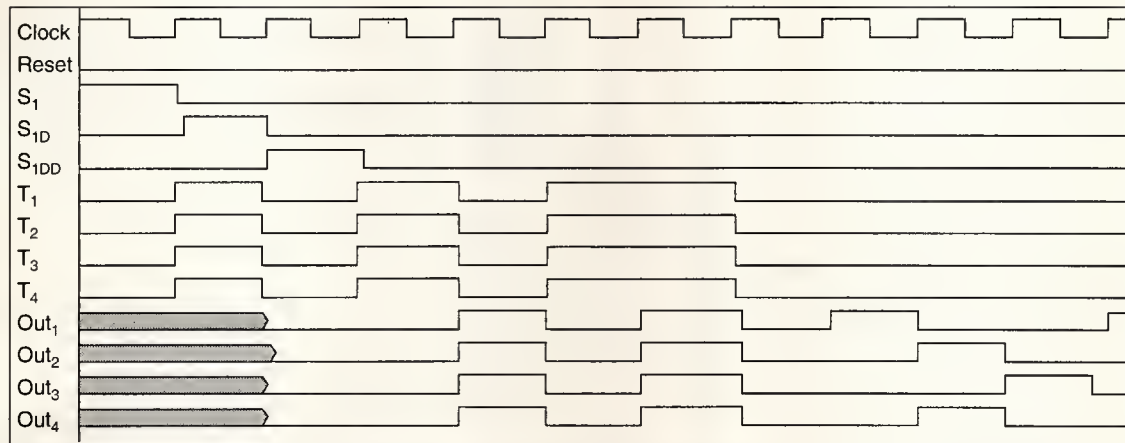


Figure 9. Simulation of the example in Figure 8.

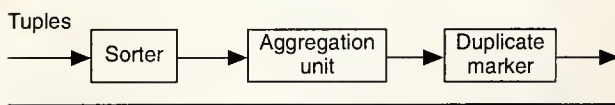


Figure 10. Processing approach of Kim et al.⁵

of the next level. We designed a four-input unit using the slice processor of Figure 4 as a basic component. Since some paths are shorter than others, a synchronization, which can be achieved through the use of delay elements, is necessary. We simulated the example in Figure 8 using this unit. The result of the simulation appears in Figure 9. Notice that the tuples are Summed, marked, and sorted concurrently. In this example, the output of the processed tuples starts three clock cycles after the input process initiation. In general, for an n -input Sum unit, the output of the processed tuples starts $\lceil \log n(\log n + 1)/2 \rceil$ clock cycles after initiation of the input process.

Comparative analysis

Kim et al.⁵ presented the design of a parallel and pipelined query processor in which tuples process in parallel, one bit at a time. Relational database aggregation functions take three steps (see Figure 10). The relation is first sorted, then fed to an aggregation unit, and the resulting relation is sent to a duplicate marker. Three different network topologies perform statistical aggregation functions.

In our design, sorting, aggregation, and duplicate marking take place concurrently. One network topology and one type of processing element (slice processor) implement these functions.

The parameters used in the comparative analysis are

- k , the number of bits in the binary representation of the

Group-By values representation;

- p , the number of bits in the binary representation of the aggregate values;
- r , the time (in seconds) to manipulate and pass one bit to the neighboring slice processor; and
- n , the number of inputs in the Sum unit.

In our implementation, the processing of the reduced tuples completely overlaps the input and output of the reduced tuples to and from the Sum unit. Since the longest path in an n -input Sum unit is $\lceil \log n(\log n + 1)/2 \rceil$ processing n tuples will take

$$A(n) = \left\lceil \frac{\log n(\log n + 1)}{2} + (k + p + 1) \right\rceil r \quad (2)$$

Notice that our approach allows for pipelined processing of different streams of tuples (different relations). (The flags of each slice processor in the first column should be initialized by applying reset signal S_1 after the input of each relation completes.) Suppose that m relations, each composed of n tuples, are to be processed by the Sum unit. The processing of these m relations in a pipelined manner will reduce the processing time from $mA(n)$ to $A(n) + (m - 1)(k + p + 1)r$. The processing is therefore reduced by

$$\left(\frac{m-1}{2} \right) \log n(\log n + 1) r \quad (3)$$

seconds.

The Sum algorithm we have described was an internal one. That is, we assumed the number of reduced tuples to be processed would be no larger than the number of inputs (n) of the Sum unit. In general, an entire relation cannot be pro-

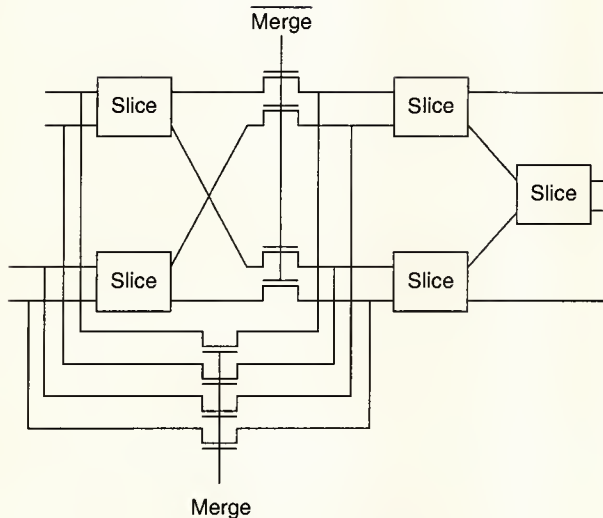


Figure 11. A Sum unit with a Merge signal.

processed internally by an odd-even network. When the number of tuples is too large for a Sum unit to process internally, an external algorithm is the most practical solution. An external algorithm is one that allows a chip (or a set of chips) of fixed size to process an input set of any size.⁴ One approach to this problem is based on an iterative use of a Sum unit of fixed size.

We based the proposed algorithm on successively merging Summed sets of reduced tuples of increasingly larger size. The external algorithm will use a Sum unit of fixed size (n -inputs) to process in an iterative manner a set of reduced tuples whose number N is larger than n . Toward this end, we added a new control signal called Merge to the Sum unit. This signal allows an n -input Sum unit to have two operating modes. In the first mode (Merge = 0), the unit performs the Procedure Sum internal algorithm just illustrated. In the second operating mode (Merge = 1) the unit merges two Summed sets of size $n/2$ each in $\log n$ steps. Figure 11 shows a four-input Sum unit with a Merge control signal.

The external algorithm is divided into two steps. During the first step the Merge signal is reset to zero, and the Sum unit generates N/n Summed sets of size n each. (Without loss of generality, N is assumed to be a multiple of n .) When the N/n sets process in a pipelined manner, the duration of this step is

$$\frac{\log n(\log n + 1)}{2} + \frac{N}{n}(k + p + 1)r \quad (4)$$

In the second step, the histogramming unit operates as a $[(n/2) \times (n/2)]$ two-way merger by setting the Merge signal to

1. The second step requires $\log(N/n)$ phases. During the i th phase, the unit converts $1 \leq i \leq \log(N/n)$, $(1/2)^{i-1}(N/n)$ sets of $2^{i-1}n$ Summed reduced tuples to $(1/2)^i(N/n)$ sets of $2^i n$ reduced tuples each. Pipelining the $\log(N/n)$ phases yields the following duration.

$$\left(\frac{\log n(\log n + 1)}{2} + \left(\sum_{i=1}^{\log(N/n)} \left(\left(2^{i+1} - 1 \right) / 2^i \right) (k + p + 1) \right) \right) r \quad (5)$$

$$= \left(\frac{\log n(\log n + 1)}{2} + \left(2 \frac{N}{n} \log \frac{N}{n} - \frac{N}{n} + 1 \right) (k + p + 1) \right) r$$

When pipelining is used between the two steps, the overall duration of the external algorithm is

$$A(n, N) = \left(\frac{\log n(\log n + 1)}{2} + 2 \left(\frac{N}{n} \right) \log \left(\frac{N}{n} \right) + (k + p + 1) \right) r \quad (6)$$

In the VLSI query processor of Kim et al., tuples are processed in a tuple parallel bit-serial fashion. Unlike our approach, this processor performs aggregation functions on presorted relations. An $(n \times n)$ sorter is used to sort internally n tuples and can also be used as a $(n \times n)$ two-way merger. Designing an $(n \times n)$ sorter requires $n(3n - 1)$ bit-serial processing elements. The longest path in the sorter consists of $(2n - 1)$ stages. The time needed to sort n tuples is $(k + p + 2n - 1)r$.

This sorter requires $4n$ I/O pins, as opposed to $2n$ for our Sum unit. The duration of merging two sorted lists of n tuples each is the same as that of sorting n tuples. The time needed to sort $N \geq n$ tuples can be computed using an approach similar to the one used to compute $A(n, N)$. First we sort each N/n set of n tuples. Next, the unit is used as an $(n \times n)$ two-way merger. As Kim et al. indicate, the overall duration of the sorting is

$$\frac{N}{n} (k + p + 2n - 1) \left(1 + \log \frac{N}{n} \right) \quad (7)$$

The sorted relation is then input to the aggregation unit. An n -input aggregation unit composed of $\log n$ stages requires $n \log n$ processing elements. Processing N tuples will require about

$$\frac{N}{n} (k + p + \log n) r \quad (8)$$

The third step is the process of assigning a mark bit to each reduced tuple so that duplicates can be removed. An n -

Table 2. $B(n,N)/A(n,N)$ for $p = k = 16$ bits.

N	$N/n=2^4$	$N/n=2^8$	$N/n=2^{10}$	$N/n=2^{12}$	$N/n=2^{14}$
16	1.50	1.20	1.16	1.12	1.10
32	2.05	1.75	1.69	1.65	1.62
64	3.50	2.96	2.85	2.78	2.72


input duplicate checker composed of $(2n - 1)$ PEs organized in two stages completes this step.⁵ Processing N tuples will require about $(N/n)(k + p + 3)r$ time.

The overall duration of performing Sum is thus

$$B(n, N) = \left[\begin{aligned} &(k + p + 2n - 1) \left(1 + \log \frac{N}{n} \right) \\ &+ (k + p + \log n) + (k + p + 3) \end{aligned} \right] \frac{N}{n} r \quad (9)$$

The comparative analysis shown in Table 2 makes it clear that the odd-even approach is significantly faster than the approach in Kim et al. For example, when $N/n = 2^{10}$, the acceleration is 1.69 for $n = 32$ and 2.85 when n is increased to 64. Also, the performance gap decreases as the ratio N/n increases. We attribute this fact to the query processor sorter/merger, which requires $4n$ I/O pins, as opposed to $2n$ for the Sum unit.

OUR SPECIAL-PURPOSE, parallel bit-level, pipelined processing unit supports relational database aggregate functions. The architecture is based on the odd-even network topology, and the system is composed of one type of simple slice processor. Each slice operates on data, bit by bit, making the design and verification of the circuit easy. The data processing time is completely overlapped with the input and output of data to and from the unit. The design is independent of the tuple size, and since a bit-serial computation is used, the system requires limited interconnection.

We designed and simulated a prototype four-input unit for fabrication using discrete components. Currently, we are designing and fabricating a VLSI prototype unit and studying clock skew over long serial paths and the compatibility of the design to RAM. 

Acknowledgments

We thank all who helped us in this project. H. Mounaf provided the design and simulation. Wayne Patterson, director of the New Orleans Advanced Computation Laboratory, gave us the opportunity to use the computing facilities of his laboratory at various stages of the project. R. Loggins helped prepare the manuscript, and the referees offered comments and helpful discussions of the issues.

Parts of this article appear in the *Proceedings of the Sixth International Workshop on Database Machines* dated June 1989 and the *International Conference on Parallel Processings* dated August 1990. A grant from the University of New Orleans Research Council and the NSF/Louisiana Stimulus for Excellence in Research, EPSCoR Program under Grant NSF/LaSER(1990)-RFAP-14 partly supported our work.

References

1. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, Vol. 13, No. 6, pp. 377-387.
2. M.S. Giovanni, "Fragmentation: A Technique for Efficient Query Processing," *ACM Trans. Database Systems*, Vol. 11, No. 2, 1986, pp. 113-123.
3. K.E. Batcher, "Sorting Networks and their Applications," *Proc. Spring Joint Comp. Conf.*, Vol. 32, Apr., 1968, pp. 307-314.
4. M. Abdelguerfi, S. Khalaf, and A.K. Sood, "Bit-Serial Parallel Processing Unit for the Histogramming Operation," *IEEE Trans. Circuits and Systems*, Vol. 37, No. 7, July 1990, pp. 948-954.
5. W. Kim, D. Galski, and D.J. Kuck, "A Parallel Pipelined Relational Query Processor," *ACM Trans. Database Systems*, Vol. 9, No. 2, June 1984, pp. 214-242.
6. A.K. Sood, M. Abdelguerfi, and W. Shu, "Hardware Implementation of Relational Algebra Operations," in *Database Machines: Modern Trends and Applications*, NATO ASI, Series F, Springer-Verlag, Berlin, 1986, pp. 321-380.

The authors' biographies, pictures, and addresses appear in the Guest Editors' Introduction on p. 7.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 165

Medium 166

High 167



A Parallel, Scalable, Microprocessor-Based Database Computer for Performance Gains and Capacity Growth

The multiback-end database supercomputer, or MDBS, consists of the architecture and performance of an experimental database computer and a number of database processors and their corresponding database stores. The author relates two studies: one on the design goals and architectural considerations of the microprocessor-based MDBS and the other on the performance expectations and benchmark results in various loads and configurations.

David K. Hsiao

Naval Postgraduate School

The MDBS architecture is unique in that each database processor is microprocessor-based and has its own database store. These stores consist of two different types of disk drives: smaller, Winchester-type drives for paging and metadata (keys and key-related data); and a larger, standard-type drive for base data (database data). The multiple database processor-store pairs, known as database back ends, interconnect through a local area network, with reliable point-to-point communications (one processor to another) and unreliable broadcast communications (one processor to many others).

The interconnected database back ends interface with another computer either directly via the LAN or indirectly via the database controller. With a direct interface, the database controller software runs in another computer known as the front end or server. With an indirect interface, the database controller is a microprocessor-based computer that uses either a tape station or built-in cassette tapes.

We use MDBS as a research vehicle in the Laboratory for Database Systems Research at the Naval Postgraduate School for the study of the design and performance of parallel and scalable database back ends. At present, MDBS can be configured or scaled into a one-back-end, two-back-end, and up to an eight-back-end database computer for parallel operations and performance analyses.

The performance gain of MDBS is unique in that the response-time reduction of a transaction is in direct proportion to the number of back ends configured. For example, if we double the number of back ends in a parallel configuration, we reduce the response time of the same transaction in the same database by almost half. Although the database remains the same in both configurations in this example, it must be redistributed to induce parallel access to any new, as well as existing database stores. We use response-time reductions to measure the performance gains of the database computer compared to the degree of its parallelism or the number of parallel back ends used.

The growth capacity of MDBS is unique in that the response-time invariance of a transaction may be nearly upheld if the increasing degree of back-end parallelism is in direct proportion to the growing capacity of the database. In other words, if we want to have nearly the same response time for a transaction as its database doubles, we simply double the number of parallel back ends. Again, we must recluster and redistribute the grown database on the new as well as existing back ends to induce parallel accesses to all the database stores. We use response time invariance to provide nearly constant response times for the same transactions, despite database growth. We simply add parallel back ends proportionally.

Finally, in this article, we point out some of

the difficulties and examine some unfinished studies that may have some impact on the architecture of parallel database back ends in general, and MDBS in particular. We have learned these lessons in the course of experimenting with this new architecture. We hope our architectural design and performance methodology will serve as a basis for the design and construction of future parallel, scalable, and microprocessor-based database computers.

The MDBS architecture

Figure 1 depicts the MDBS architecture. The major building block for the high degree of parallelism is the database back end. To achieve a high degree of parallelism, we simply add identical back-end hardware to the LAN, replicate the back-end software and metadata on the new hardware, recluster as necessary, and redistribute the base data onto the new and existing database stores. Let us elaborate on the major building block first. We can then discuss the rest of the architectural elements.

Database back ends. Eight database back ends are available in our laboratory. Although we are not specific in Figure 1, all eight back ends are identical. We therefore focus on the design of a single back end in the following sections.

Hardware elements. Each database back end is a combination of a database processor and a database store.

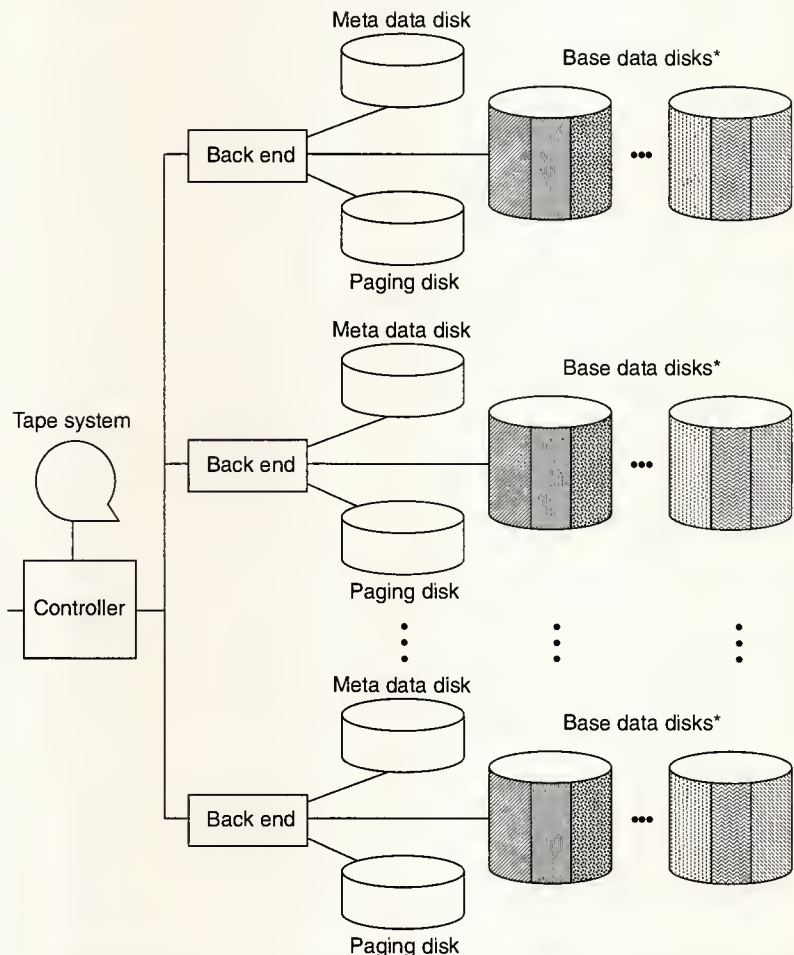
- **CPU and internal data bus.** Each database processor consists of a Motorola 68020 microprocessor-based CPU with internal 32-bit-wide data and control buses. At 16.67 MHz, the 68020 is adequate for database operations, since the operations are mostly I/O-intensive rather than computationally intensive. Further, for any data transfer between the real memory and external buses, we use the VMEbus. The VME bandwidth is 15.5 Mbytes/s. On the other hand, the real memory cycle of the database processor is 240 ns per 32-bit word. We use the following calculation:

$$\begin{aligned} 4/240 \text{ bytes/ns} &= 4 \times 10^9/240 \text{ bytes/s} \\ &= 16.67 \times 10^5 \text{ bytes/s} = 16.67 \text{ Mbytes/s.} \end{aligned}$$

Thus, since 15.5 Mbytes/s is below its 16.67-Mbytes/s capability, the maximum amount of data that can be concurrently transferred in (and out) of real memory

for processing over four buses is nearly covered by the bandwidth of the VMEbus. The buses consist of three for I/O—paging, metadata, and base data—and one communications bus for the LAN. There are still some memory cycles left (at the rate of 11.17 Mbytes/s) for the CPU control bus to access the real memory.

- **Backplane and communication bus.** All four external buses do not need to access the VMEbus simultaneously. On those rare occasions when competition among memory cycles for real-memory access is keen, the CPU control bus always gives the right-of-way to I/Os and communications. Further, the communications bus, known as the backplane of the back end, is managed by another microprocessor with its own buffer memory, and also gives the right-of-way to I/O buses.



* Tracks with the same shading consist of records belonging to the same cluster.

Figure 1. Architecture of MDBS.

We discuss the backplane hardware later.

- **External I/O buses and the internal VMEbus.** As discussed, the I/O buses can monopolize the entire bandwidth of the VMEbus. The microprocessor-based CPU and real memory can thus sustain disk drives whose maximum transfer rate is about 16 Mbytes/s.
- **Disk drive types.** Our system contains three disk drives; one is for paging, another for metadata, and the third for base data. Because the 68020 microprocessor supports address translation, a larger virtual memory (from 2 Mbytes to 32 Mbytes) may be supported by a smaller, 4-Mbyte real memory. There is still, however, need for a paging disk for virtual memory pages. MDBS uses a 96-Mbyte Winchester-type disk drive for paging, because most of the operating and database system modules are real memory-resident and their pages are locked into the real memory. We use another 96-Mbyte disk drive for the metadata. You will recall that database metadata are keys and key-related information. Queries use keys to access clustered records directly. The number of keys used relates to the average size of the database clusters, which in turn relates to the number of back ends in the configuration. Finally, a large, standard, 500-Mbyte moving-head disk stores the base data. Here, the key-to-record ratio of metadata storage to base data storage is 96:500, nearly 1:5 per back end.
- **Disk transfer rates.** The transfer rate of each disk drive is under 2 Mbytes/s. Thus, with three drives transferring simultaneously in or out of real memory, the maximum rate is under 6 Mbytes/s. On the other hand, real memory can sustain a rate of some 16 Mbytes/s, so there is enough capacity for the addition of, for example, two more sets of metadata and base data disks.
- **Database store.** For each back end there may be 288 Mbytes of metadata and 1.5 Gbytes of base data for its database store. For the eight-back-end database computer in our laboratory, there is a potential for some 12 Gbytes of database data. On the other hand, the metadata are replicated. Thus, the real capacity of metadata remains at 288 Mbytes. The key-to-record ratio drops from 1:5 in the one-back-end configuration to nearly 1:40 in the eight-back-end configuration. To maintain the capacity of metadata relative to the capacity of base data, we may add more 96-Mbyte metadata disks and fewer 500-Mbyte base data disks. In other words, the database store is scalable for direct access, depending on the capacity of the metadata and base data. The database back end is also scalable. In our laboratory, it spans from one to eight.

Software processes. Since a database is considered as residing in a back end, we use the Directory Management process to manage metadata. We also use a process for managing

base data, the Record Processing process. In each back end, transactions execute concurrently. Metadata and base data processing of a transaction take place serially, and the metadata processing of a transaction may overlap with the base data processing of another transaction. And, since metadata processing as well as base data processing for different transactions may take place concurrently, we need a Concurrency Control process.

A back end must be able to receive transactions for processing and return responses. Thus, each back end has a pair of processes for communications: the Get process, for getting transactions or messages from the LAN, and the Put process, for placing responses or messages on the LAN. Directory Management, Record Processing, Concurrency Control, Get, and Put are the only processes of a back end. These processes are the same in every back end in a multiback-end configuration. Further, these five database management and communication processes use a Unix BSD 4.3 operating system with TCP/IP protocols.

A microprocessor-based LAN. Through its Get and Put processes, the microprocessor-based LAN is the only communications link between the controller and back ends.

LAN hardware elements. These elements are an Ethernet cable, the backplanes, and their transceivers. Each back end contains a backplane that consists of an Intel 80186 microprocessor and a small amount of real memory. The 128-Kbyte real memory is used mainly for buffering incoming or outgoing messages. It is also dual-ported to another microprocessor (Intel 82586) that mainly executes low-level protocols, which in turn support high-level TCP/IP protocols. These protocols in turn support even higher protocols, such as UDP (Ethernet's user-defined protocol).

The Ethernet cable connects all the backplanes by way of transceivers. It thus enables all 82586 microprocessors to work in concert to execute a collision-resolution algorithm whenever there is a collision or conflict of messages. One of the messages is then selected for routing, while all other messages are deferred momentarily and considered for routing again. If, in the next routing, there is another collision, the algorithm executes again. Eventually, all messages will be routed to their designations so no messages are lost.

This algorithm causes restricted point-to-point (one backplane to another backplane) routing. In other words, if, for example, three point-to-point routings collide (say, from Backplane 1 to Backplane 4, from Backplane 3 to Backplane 8, and from Backplane 7 to Backplane 2), the three pairs route one after another, although we cannot predict the routing sequence. We note that, in this example, all backplanes involved in the routing are disjointed, and no two pairs overlap on an identical backplane. The Ethernet cable has a transmission rate of 10 Mbps.

On the other hand, the 82586-based Ethernet does not provide reliable broadcasting (one backplane to many

backplanes routing). For example, if Backplane 2 wants to broadcast a message to all other backplanes, namely, Backplanes 1 and 3 through 8, the routing takes place. However, the message may not get to one or more of the other backplanes, let's say, Backplane 7. This kind of unreliable routing is also unpredictable. Lastly, the 82586-based Ethernet does not provide reliable multibroadcasting either.

LAN software protocols. Since the LAN's own TCP/IP software does not support reliable broadcasting and multibroadcasting, we built another layer of protocols, UDPs, for these operations.

At each back end, a logical socket for each and every other back end is defined. Whenever a message reaches a particular back end, that back end must acknowledge receipt. The acknowledgment is deposited in its corresponding socket at the sender's back end. This send-receive-acknowledge procedure via a socket is not necessary if the message routing is point-to-point (one back end to another). In other words, the acknowledgment is always there and can be ignored.

***Our investment in the software
is minimal, because it involves
only a few memory locations
for sockets.***

In broadcasting a message to other back ends, the UDP software examines the acknowledgments deposited at the sender's back end. If there is no acknowledgment at a specific socket, the program resends the message to the corresponding back end's socket by way of point-to-point routing. Obviously, the receiver's back end cannot begin its database operation simultaneously and in parallel with other back ends that had an earlier start on the same message. MDBS's back ends are not tightly coupled, do not share primary and virtual memories, and have their own database stores. Thus, all other back ends can proceed individually and in parallel without any interference or delay from one or more identical back ends.

The UDP software helps make unreliable broadcasting reliable. Our investment in the software is minimal, because it involves only a few memory locations for sockets. The use of existing and reliable point-to-point protocols for resending messages is overhead, but it does not interfere with other back ends that require no resending. So, with reliable broadcasting, we now also have reliable multibroadcasting—a communication feature necessary to our back ends. The back ends

and controller access the UDP software with Get and Put.

Database controller. As stated previously, the controller software may run in a general-purpose front-end computer that interfaces with the database back ends via the LAN. However, because it does not interfere with other program activities and programs in the front-end computer, for research and experimental purposes, a dedicated database computer like MDBS is preferred.

Controller hardware. Like a back end, the controller hardware is based on a 68020 microprocessor with address translation capability, so we still need a paging disk. Unlike a back end, the controller accesses neither metadata nor base data. On the other hand, the controller is responsible for the backup and recovery of the back end and LAN software. Thus, we place a tape station at the controller. Backup and recovery tapes for on-line and real-time backups and recoveries can be readily made. We also plan to use tapes to load a massive new database into back-end database stores. In addition, we use tapes to load benchmarks for performance analyses.

Controller software. Because the controller computer is the sole interface with the "outside" world, it needs two processes: one for receiving database transactions from the front end and another for returning database results to the front end. These processes are known as TP (Request or Transaction Processing) and PP (Postprocessing).

A user request may be a new database transaction requiring compilation, a canned transaction or macro in a transaction library, or an on-line query requiring immediate response. TP must be able to uniquely identify each request, preprocess it, and broadcast the request to all back ends. Each back end, on the other hand, places a broadcast request in its request (or transaction) queue. The preprocessing of each transaction amounts to a translation of the transaction into one or more back-end primary database operations. We discuss the five primary database operations, RETRIEVE, DELETE, INSERT, UPDATE, and RETRIEVE COMMON, in a later section on databases and their operations.

The PP process, on the other hand, is the postprocessing of database records. For example, if a user desires the sum of all the salaries from the salary database, each back end returns a subsum of all the salaries from its portion of the salary database. Then, PP adds all the subsums to produce the overall sum that is then routed to the user. In this example, PP performs the aggregate function, Sum. In general, PP performs a large number of aggregate functions. By interfacing with TP, PP can uniquely identify the correct user to return the result or error message.

TP and PP interface with the outside world. We use an Insert Information Generator, or IIG, process designed solely to assist a back end during an INSERT operation in the "inside" world of the database computer. Of the five primary back-end database operations, the INSERT operation is the

Upon insertion of a record, the particular cluster in which the record belongs must be identified. However, the cluster is evenly distributed over a number of back-end disk stores. The database store that provides the latest available storage space for any record insertion is identified in a space utilization table maintained by the IIG. The space utilization table keeps the following information up to date. For each cluster of records, it identifies the back end whose database store contains the first trackful of the cluster, and it identifies the back end whose database store contains the last trackful of records. It also identifies the back end whose database store can provide the first available track for inserting the new trackful of clustered records. The maintenance and allocation by the IIG are based on a round-robin algorithm explained later. In any case, with the help of the space utilization table, the IIG instructs a specific back end to insert records into its database store. The actual insertion is made through the processes of the specific back end. Further, since metadata appear in every back end, the specific back end must also broadcast its metadata updates to all other back ends for replication.

Architectural elements and system processes. In Figure 2, we illustrate the relationship between the elements and the system processes. Since all the architectural elements and system processes in a back end are identical to the ones in another back end, we depict those in only one back end. We observe that, for communications, the Put process of one computer sends messages to one or all Get processes in the other computers. Thus, Put can facilitate either one-to-one or broadcasting in its communication with the other computers in the system.

Figure 1 illustrates the architecture of the system, divided into two main components: the Front-end computer and the Back end, connected via a LAN.

Front-end computer:

- TP (Transaction Processing):** Receives data from the 'From' source and sends it to PP and Put.
- PP (Postprocessing):** Receives data from TP and sends it to Get and back to TP.
- IIG (Insert information generator):** Connected to both Put and Get.
- Put (Put module):** Receives data from TP and IIG, and sends it to the LAN.
- Get (Get module):** Receives data from the LAN and IIG, and sends it to PP.

Back end:

- Get (Get module):** Receives data from the LAN and sends it to CC.
- CC (Concurrency Control):** Receives data from Get and sends it to DM and RP.
- DM (Directory Management):** Receives data from CC and sends it to RP.
- RP (Record Processing):** Receives data from CC and DM, and sends it to Put.
- Put (Put module):** Receives data from RP and sends it to the LAN.

Data Stores:

- Meta data store:** Connected to DM.
- Base data store:** Connected to RP.

Legend:

- CC Concurrency Control
- DM Directory Management
- IIG Insert information generator
- PP Postprocessing
- RP Record Processing
- TP Transaction Processing

Data organization and repertoire of operations

- models metadata and base data separately;
- introduces an equivalence relation, which partitions the database into mutually exclusive sets of base data called clusters; and
- allows clustered records to be distributed in back ends, which induces parallel access to database stores.

The rationale for implementing ABDL instead of a conventional data language such as SQL, derives from the following intrinsic properties of ABDL. ABDL

- supports a parallel search algorithm based on predicates, and
- is semantically rich and complete so that transactions written in conventional data languages like SQL may be translated into ABDL for execution in MDBS.

These and other properties of ABDM and ABDL are discussed later.

Base data and metadata. Every piece of data in the database is characterized in ABDM as an attribute-value pair. Thus, when we refer to a piece of data, say, USA, in an example database, we not only refer to its value, USA, we also know its attribute, Country. An attribute-value pair is formally denoted as an ordered pair enclosed in angular brackets, <Country, USA>. Attribute-value pairs are the building blocks of the database.

Base data. A record is a set of attribute-value pairs such that no two attribute-value pairs of a record have the same attribute, and at least one attribute of a record is typed. The first rule ensures that any attribute of the record is single-valued. The second rule ensures that at least one key identifies the record. A record is formally denoted as a set of attribute-value pairs enclosed in parentheses: (<File, aircraft>, <Classification, top-secret>, <Plane-Type, fighter>, <Radius, 799 nautical miles>, <Country, USA>, <Fuel-capacity, 600 gallons>, <Pilot-Grade, test-pilot>). All the records of the database comprise its base data. In reality, there are thousands, if not millions, of base data or records, in the database.

Metadata. There are three attribute types. The Type A attribute is one of disjointed value ranges. Thus, a Type A attribute partitions its values into value ranges. For example, the attribute Radius in Table 1 is a Type A. A Type B attribute is one of distinct values. For example, the attribute Country in Table 1 is a Type B, since it has two distinct values, USA and USSR (shown in Table 2). A Type C attribute is one of those distinct values being entered by the user in real time. Thus, Type Cs are like Type Bs in assuming distinct values. However, Type B attributes are created at the time of database creation or at generation time. Attributes and their types collect in the Attribute table (Table 1). For a real-world database, an attribute table consists of tens, if not hundreds, of attributes. We illustrate only five in Table 1.

We call each Type A attribute and its value ranges Type A descriptors. Similarly, we call Type B and Type C attributes and their distinct values Type B and Type C descriptors. If we relate the descriptors here to conventional keys, we have three different kinds of keys. Whereas both Type A and Type B descriptors tend to remain the same, because of their creation at database generation time, the Type C descriptors

may increase rapidly if a user enters records consisting of many new values having the Type C attribute. All the descriptors collect in a table known as a descriptor-to-descriptor-identifier table, or for short, the Descriptor table (Table 2). For a real-world database, a descriptor table consists of hundreds, if not thousands, of descriptors. In Table 2, we illustrate only 16. Specifically, Radius results in six Type A descriptors; Plane-Type, in three Type C descriptors; Country, in two Type B descriptors; File, in one Type B descriptor; and Classification, in four Type C descriptors.

Let D_i be the set of $D_{i,j}$ for all j . Then the product $D_1 \times D_2 \times \dots \times D_n$ is an equivalence relation whose members partition the database data into mutually exclusive sets of records, or attribute-value sets. These record sets we term clusters.

Referring to our sample in the Attribute and Descriptor tables, we note there are five attributes. Thus, $n = 5$. For D_1 , or Radius, we see six descriptors. Thus, for $i = 1, j_i = j_1 = 6$. Similarly, for $i = 2, j_2 = 3$ for $D_2 = \text{Plane-Type}$; for $i = 3, j_3 = 2$ for $D_3 = \text{Country}$; for $i = 4, j_4 = 1$ for $D_4 = \text{File}$; and for $i = 5$,

Table 1. Attributes.

Attribute	Attribute type	DDIT entry
Radius	A	D11
Plane-type	C	D21
Country	B	D31
File	B	D41
Classification	C	D51

Table 2. Descriptors.

ID	Descriptor
D11	$0 \leq \text{Radius} \leq 400$
D12	$401 \leq \text{Radius} \leq 600$
D13	$601 \leq \text{Radius} \leq 800$
D14	$801 \leq \text{Radius} \leq 1,000$
D15	$1,001 \leq \text{Radius} \leq 1,200$
D16	$1,201 \leq \text{Radius} \leq 2,000$
D21	Plane-Type = fighter
D22	Plane-Type = bomber
D23	Plane-Type = reconnaissance
D31	Country = US
D32	Country = USSR
D41	File = aircraft
D51	Classification = top-secret
D52	Classification = secret
D53	Classification = confidential
D54	Classification = unclassified

$j_5 = 4$ for $D5 = \text{Classification}$. The cardinality of the product $D1 \times D2 \times D3 \times D4 \times D5$ is 144, since $j_1 \times j_2 \times j_3 \times j_4 \times j_5 = 6 \times 3 \times 2 \times 1 \times 4 = 144$.

Potentially, this database may have up to 144 clusters, each of which is uniquely identified by a set of descriptor identifiers, with corresponding descriptors characterizing the cluster records. In reality, many clusters are empty, since there are no records in the database being interpreted by the descriptor identifier sets determining the clusters. MDBS does not track empty clusters. Nor does it record their descriptor identifier sets in the Descriptor Table. The three kinds of identifiers are the

- 1) system-generated *record*, which identifies the records of a cluster characterized by a descriptor identifier set;
- 2) *cluster*, which identifies the cluster containing the records; and
- 3) *descriptor*, which identifies the descriptor-identifier set that determines the cluster.

These identifiers reside in the table known as the database cluster definition table, or, for short, the Cluster table. In our sample database, the Cluster table (Table 3) identifies 12 clusters of 33 records. Each cluster is defined by a unique set of descriptors. For example, the first cluster C1 is defined by five descriptors whose identifiers are D13, D21, D31, D41, and D51. Three records in that cluster have their own record identifiers, R1, R2, and R13. Despite their differences, these records all consist of top-secret aircraft data (defined by D51 and D41) about US fighters (identified by D31 and D21) hav-

ing a combat range of 601 to 800 nautical miles (D13). In practice, a cluster table has tens, if not hundreds, of entries.

The three database tables, Attribute, Descriptor, and Cluster, contain the database metadata. Since the product $D1 \times D2 \times \dots \times Dn$ induces an equivalence relation, no record identified in the Cluster table can belong to two different clusters. Furthermore, records in each cluster are unique. The equivalence relation is the foundation for our database parallel access and storage strategies.

Distribution. The distribution of metadata and base data to their separate database stores takes place differently, although both types attempt to access their stores in parallel.

Metadata. Since metadata is typically one or two orders-of-magnitude smaller in size than base data, we decided to replicate the metadata onto each of the back-end database stores. We also use the smaller, dedicated disk drives for storage of replications. Subsequently, all the back ends can access their own metadata stores and the same sets of Attribute, Descriptor, and Cluster tables, in parallel.

Furthermore, parallel metadata accesses may be overlapped with the parallel base data accesses for any other transaction, since base data stores use separate and larger disk drives. Therefore, MDBS achieves concurrent and parallel executions of transactions.

Base data. Base data make up the bulk of a database. Therefore, they are not replicated for storage. Further, they are stored on each back end's own high-capacity disk drives in a prescribed fashion. The method is as follows:

- 1) from the Cluster table, the controller picks up a cluster identifier and its associated records;
- 2) the controller blocks a variable-size cluster into fixed-size trackfuls of records;
- 3) the controller determines the identifiers of the back ends, each of which can provide a track of available storage for the cluster identified (recall that the controller IIG has a storage utilization map to keep track of such information);
- 4) the controller sends in parallel all the trackfuls of records to the back ends identified;
- 5) each identified back end places its block of one or more trackfuls of clustered records into its base data store and enters identifiers of records stored onto the replicated Cluster table entry corresponding to the cluster on the metadata store;
- 6) the controller then updates its space utilization map with respect to this cluster; and
- 7) the entire procedure is repeated for all subsequent clusters.

This turn-taking and one-track-per-back-end database distribution (or redistribution) algorithm has a desirable effect in that records of a cluster are evenly distributed (or redistrib-

Table 3. Clusters.

ID	Descriptor ID set	Record ID
C1	{D13, D21, D31, D41, D51}	R1, R2, R13
C2	{D12, D21, D31, D41, D51}	R8, R9
C3	{D14, D21, D32, D41, D51}	R20
C4	{D13, D21, D32, D41, D51}	R17
C5	{D12, D21, D32, D41, D51}	R21, R24, R27, R30, R31
C6	{D13, D21, D31, D41, D52}	R3, R4, R14
C7	{D12, D21, D31, D41, D52}	R10, R11
C8	{D13, D21, D32, D41, D52}	R18, R19
C9	{D12, D21, D32, D41, D52}	R22, R23, R25, R26, R28, R29, R32, R33
C10	{D13, D21, D31, D41, D53}	R5, R6, R15
C11	{D12, D21, D31, D41, D53}	R12
C12	{D13, D21, D31, D41, D54}	R7, R16

uted) over a set of separate, parallel database stores. Subsequent accesses to the records of a cluster become parallel. In other words, we have induced the record-parallel and cluster-serial, or RPCS, operation for our database access operations.

The choice of the cluster size is important in a parallel access operation. Let the number of record tracks in a cluster be m and the number of back ends n . Then, the maximum number of record tracks for a cluster should also be n , for example, $m < n$. To fine tune the cluster size, we control the number of descriptors used in the identification of the clusters. In general, the more descriptors used, the smaller the clusters become. If MDBS has many n and few m , say, $3m = n$, its back ends can access several (three) clusters in parallel. Thus, we have induced record-parallel and cluster-parallel, or RPCP, operation for our database access operations. Figure 1 depicts both RPCS and RPCP operations for a sample distribution of clusters of the database.

Database operations. A database transaction is composed of one or more of the five primary database operations, RETRIEVE, DELETE, INSERT, UPDATE, and RETRIEVE COMMON (see Figure 3). Except INSERT, which inserts one new record at a time into an existing database, these operations are set-oriented. More specifically, RETRIEVE, DELETE, and UPDATE operate on one set of records at a time, whereas RETRIEVE COMMON operates on two sets of records at a time.

The query—a Boolean expression of predicates. The input for all set-oriented primary database operations is a query. The output of an operation is a set of records that has satisfied the query and has been operated on. The query allows us to specify the properties of a record set without having to provide the record addresses of the set. These properties are specified in terms of a Boolean expression of predicates. A predicate is a 3-tuple, consisting of an attribute, a relational

operator, and a value. The set of relational operators include $=$, $<$, $>$, \leq , and \geq . Boolean operators include \wedge (And), and \vee (Or). The following is a query consisting of the conjunction of two predicates (one less-than-or-equal-to predicate and one equality predicate) and one disjunction of three equality predicates.

(((Radius \leq 600) \wedge (File = aircraft) \wedge ((Classification = secret) \vee (Classification = confidential) \vee (Classification = unclassified))))

Transaction processing in the controller transforms all queries into their equivalent disjunctive normal forms for set-oriented primary operations.

Parallel predicate search algorithm. In Figures 4 and 5 on the next two pages, we depict our parallel search-for-records algorithm on the basis of a given query. To simplify the illustration, we use a simple query as a conjunction of two predicates instead of the disjunctive normal form of many predicates. Given a query, the back ends can determine the descriptor sets that satisfy the predicates of the query, since descriptors and predicates are similar in form. They do differ in where they appear. Descriptors are held in metadata or Attribute and Descriptor tables, and predicates in user queries. Once determined, we use these descriptor sets to compute the descriptor-identifier sets, which in turn allow us to determine the clusters characterized by them in the Cluster table. Thus, at the time the parallel search algorithm is being used, two input parameters have been given to the algorithm: the query and the cluster identifiers from which records will be searched for checking against the query.

Let us observe the algorithm with the following sample query in Figure 4:

- 1) The controller broadcasts the query to all the back ends for execution.
- 2) Each back end uses the query to determine those clusters that may have records satisfying the query. Each back end's Descriptor and Cluster tables aid this determination. The cluster number and identifiers determined for the query are now known by all back ends.
- 3) All back ends access one or more clusters of records. Since clustered records are evenly distributed over the back ends' disks and the accesses are either RPCS or RPCP operations, parallel database streams flow to the back ends from their respective database stores.
- 4) Each back end does the following: It compares the attribute of the first predicate of the query with the attribute of the first attribute-value pair of the forthcoming record. If the attribute matches, the back end checks whether its attribute value satisfies the value of the predicate dictated by the relational operator of the predicate. If satisfied, it repeats the comparison and checks for the

RETRIEVE (query) [target-list] [by-clause]

DELETE (query)

UPDATE (query) (modifier)

RETRIEVE (query-one) [target-list-one]

COMMON (attribute-one, attribute-two)

RETRIEVE (query-two) [target-list-two]

INSERT (<attribute-first, value-first>, <attribute-next, value-next>, . . . , <attribute-last, value-last>, {any arbitrary string})

Figure 3. Five primary operations.

next predicate and attribute-value pair until there are no more predicates in the query. In this case, the record satisfies the query and is routed as output. If any one of the following cases takes place, the record is not output, and the search begins with the next record in the stream. These cases are: the first comparison for identical attributes fails; the next comparison fails the relational operator test; or the record under consideration runs out of attribute-value pairs before the query runs out of predicates for comparison.

The parallel search algorithm makes one sweep of all the database streams coming from the database stores without the need to recall those records having been swept. Our encoding technique makes this possible. First, we use attribute identifiers, as illustrated in the Attribute table, in lieu of attribute names in both attribute-value pairs and predicates. We then order the attribute-value pairs monotonically by their attribute identifiers and query predicates.

The effect of this algorithm is the creation of a single-query/multiple-database-streams, or SQMD, operation. If, at a back end's database store, there is no clustered data for a query, the back end chooses the next transaction in its transaction queue to execute. The query of the new transaction is also new. Thus, we achieve a multiple-query/multiple-database-streams, or MQMD, operation, too.

MQMD operations are the best we can achieve in parallelism for a parallel database computer. However, it should be observed that the letter "Q" in SQMD and MQMD replaces the letter "I" (instruction) in SIMD and MIMD, which are the two best parallel operations in a supercomputer. Note also that the letter "D" in SQMD and MQMD, differs from the letter "D" in SIMD and MIMD. Our "D" stands for database streams coming from their respective base-data disks, while the classical "D" stands for data items coming from main memories. To sustain parallel database streams, we cluster data and distribute the clustered data evenly across multiple disk stores. Our parallel algorithms achieve

(a)

Employees	1	2	3	4	5	6
	Employee number	Employee name	Department name	Salary	Vacation earned	Vacation used

(b)

1.	1	2	3	4	5	6
	10	Smith	Sales	10000	10	0
2.	1	2	3	4	5	6
	15	Jones	Sales	30000	5	6
3.	1	2	3	4	5	6
	21	Brown	Purchasing	22000	12	0
4.	1	2	3	4	5	6
	35	Smith	Marketing	30000	9	2
5.	1	2	3	4	5	6
	42	Smith	Purchasing	20000	8	5
6.	1	2	3	4	5	6
	50	White	Purchasing	25000	0	0
7.	1	2	3	4	5	6
	62	Gray	Marketing	10000	4	2
8.	1	2	3	4	5	6
	71	Hall	Sales	15000	10	3
9.	1	2	3	4	5	6
	75	Green	Sales	10000	8	9

Field

Figure 4. Parallel search algorithm by predicates: record template (a); comparison of bit serial and database stream parallel and single sweep (b).

SQMD and MQMD, as well as RPCS and RPCP operations over the database.

RETRIEVE COMMON. As mentioned earlier, RETRIEVE, DELETE, and UPDATE are three database primary operations, each of which operates on one set of records at a time. A user query specifies the set of records to be operated on. On the other hand, RETRIEVE COMMON operates on two sets of records, each of which is specified by a query. Thus, a RETRIEVE COMMON operation involves two queries. Es-

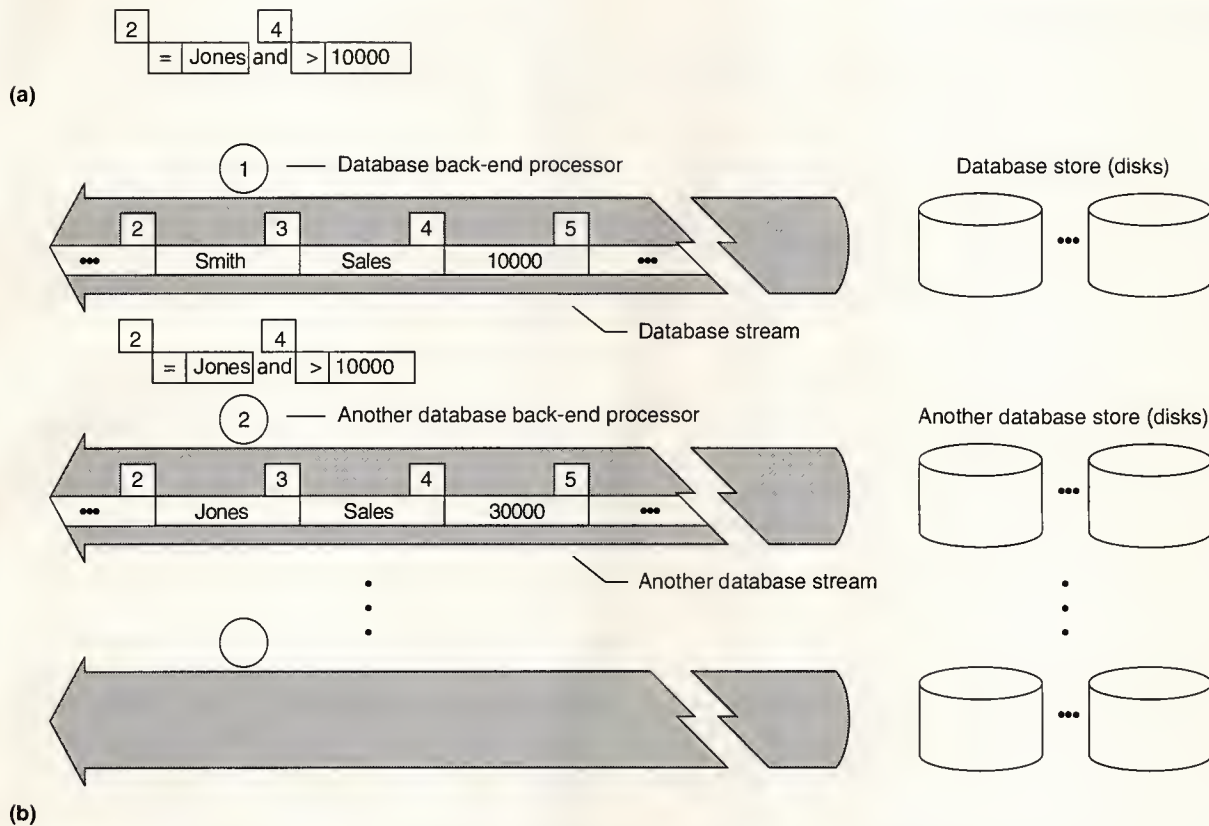


Figure 5. Conjunction of an equality predicate and a greater-than predicate (a); and predicate engine (b).

essentially, this operation retrieves two sets of records and outputs those records that have common attribute values. It is equivalent to a relational equijoin operation whose complexity is about the same as that of a tape-oriented merge of two files. However, in a parallel architecture using a broadcasting LAN, we need a unique algorithm.

The RETRIEVE COMMON algorithm for multiple back ends interconnected by a broadcasting LAN is as follows:

- 1) In SQMD mode, each back end retrieves its first subset of records on the basis of the first query given.
- 2) For each record retrieved, each back end identifies the common attribute specified in the primary operation, hashes its attribute value into a virtual memory address, stores the record in the virtual memory so addressed, and repeats this step until all the records retrieved have been hashed into the back end's virtual memory.
- 3) In SQMD mode, each back end then retrieves the second subset of records on the basis of the second query given.
- 4) For each record retrieved, each back end identifies the

common attribute as specified in the primary operation; hashes its attribute value into a virtual memory address; fetches all records with the same virtual address, if any, from the virtual memory; compares its attribute value with the attribute value of the records fetched; outputs both records when they do compare; and repeats this step until all the records of the second subset have been retrieved and processed.

- 5) Each back end then broadcasts its second subset of records to all the other back ends.
- 6) For each record received via broadcasting, each back end repeats step 4. When step 4 is finally completed for all records, the operation ends.

We must note that, although there are two sets involved in the operation, only the second set was broadcast. If there are n back ends, we have, at most, n mutually exclusive subsets of the second set, each of which is broadcast to the other $(n - 1)$ back ends. The load of the broadcast LAN is therefore the cardinality of the second set, that is, the sum of the cardinalities of its subsets.

MDBS performance

We measure MDBS performance in two ways: response time reduction, or RTR, and response time invariance, or RTI.

If the response time of a transaction is too long for our liking, we may add more database back ends, replicate both the system software and metadata on the new hardware, and redistribute the same base data onto existing and new stores. Ideally, the reduction in response time is inversely proportional to the hardware added. For example, if we double the number of back ends for a transaction in a database, we would like a reduction in response time of one half. This is the RTR.

If the response time gets longer as the database grows in size, we may add more hardware, replicate the software and metadata as above, and redistribute the increased base data onto the existing and new hardware. Ideally, the response time of the transaction will remain the same as long as the new hardware added to the configuration is in proportion to the growth of the database, say, a doubling of the number of back ends for every doubling of database size. This is the RTI. See Demurjian¹² for formal definitions and formulas of RTR and RTI.

Benchmark transactions. As an example, let's use a set of benchmark transactions and a sizeable benchmark database to illustrate RTR measurements. The database is redistributed on a variety of back-end configurations, ranging from the baseline configuration (one-back-end), to a highly parallel configuration (eight-back-end). (Refer to Figure 1.)

Let's also introduce a set of benchmark databases whose sizes are proportional to storage capacities and the number of back ends. We use the set of benchmark databases and the same set of benchmark transactions to conduct RTI measurements over eight configurations as well. There are two kinds of benchmark transactions: overhead-intensive and data-intensive.

Overhead-intensive transaction. This transaction tends to access a small portion of a database and perform intended database primary operations on even smaller portions of accessed data. Let's say that an overhead-intensive transaction (Transaction 1) is a RETRIEVE operation that accesses only 4 percent of the benchmark database, checks them against the query, and outputs only those satisfying the query. The output is about one half the data accessed, or 2 percent of the database. Let's use another overhead-intensive transaction (Transaction 6), a DELETE operation. Like Transaction 1, it accesses 4 percent of the database and qualifies one half, or 2 percent of the data accessed. Unlike Transaction 1, though, it deletes those qualified 2 percent from the database.

Data-intensive transaction. This transaction tends to access a large portion of the database and perform the intended operation on a relatively small portion of the accessed data. Let's consider three retrieves that are data-intensive. Transaction 2 accesses 26 percent of the database,

of which 96 percent of the records (25 percent of the database) satisfy the query. Transaction 3 accesses 50 percent of the database, of which one half of the records (25 percent of the database) satisfy the query. Transaction 4 accesses 100 percent of the database, of which one half of the records (50 percent of the database) satisfy the query. Two DELETES are also data-intensive. Transaction 5 accesses 50 percent of the database, with one half of the records (25 percent of the database) being deleted. Transaction 7 accesses 100 percent of the database, with one half of the records (50 percent of the database) being deleted.

We have now proposed seven benchmark transactions, where two are overhead-intensive and five are data-intensive. (Typical database operations are mostly data-intensive.) We have restricted these seven transactions to two primary database operations, RETRIEVE and DELETE. The three remaining primary database operations, INSERT, UPDATE, and RETRIEVE COMMON, are not considered for benchmarking.

Because INSERT is not a set operation, it cannot take advantage of the parallelism of our database computer. There should be no response time reduction or response time invariance over the conventional sequential database computer.

UPDATE can be viewed as a series of three operations: RETRIEVE a set of records for updating, DELETE the original set of records from the database stores, and INSERT one at a time all newly updated records.

Because RETRIEVE and DELETE have been included in the benchmark study, and INSERT has been left out, there is no need to benchmark the update function.

Finally, we have also decided not to benchmark RETRIEVE COMMON, which is not only a set operation but also involves two sets of records. At present, RETRIEVE COMMON works well with two small sets of records. They are too small to accommodate the sizes of our two benchmark record sets. The size limitations are not attributed to the reliability of the broadcast protocols, nor are they attributed to the parallel and broadcast algorithms of RETRIEVE COMMON outlined above. They are attributed to the microprocessor-based backplanes of the Ethernet. We discuss hardware issues later.

Benchmark databases. The choice of database, cluster, and record sizes for our benchmark study is the first step toward the creation of the benchmark databases.

Multiplicative factor. For the RTR study the same database is benchmarked over one to eight back ends. Thus, the database size must be divisible by each back-end configuration. We must choose, therefore, the lowest common multiple of the back-end numbers as the factor for the database size. Thus, the lowest common multiple of 1, 2, 3, 4, 5, 6, 7, and 8 is 840. For the RTI study we replicate the same database on additional back ends. Thus, for this study the total database size is the number of back ends times the size of a database.

Record sizes. There are three record sizes: 1,000 bytes for a large record, 500 bytes for a medium-size record, and 100

bytes for a small-size record. There are three attribute types: one Type A and two Type Bs. The attribute values take only a small number of bytes in each record. The rest of the record consists of filler attribute values whose attributes are not typed or kept in the metadata tables. The value ranges of the Type A attribute and the distinct values of the two Type B attributes are carefully chosen to induce the following cluster sizes.

Cluster sizes. For convenience in our benchmark effort, we chose the same byte size for each of the four different cluster sizes, although the record numbers in these clusters may be different. The four cluster sizes are large (1,000-byte records), medium-large (500-byte records), medium (200-byte records), and small (100-byte records). Each cluster contains 1.68 Mbytes.

More specifically, there are 140 large clusters, ranging from four 1,000-byte records per cluster to twenty 1,000-byte records per cluster. Together, these 140 clusters contain 1,680 large records, although some contain as few as four large records each, and others contain as many as 20 records each. Similarly, 140 medium-large clusters range from eight 500-byte records per cluster to forty 500-byte records per cluster. The medium-large clusters contain, collectively, 3,360 medium-large records. There are 140 medium clusters, ranging from twenty 200-byte records per cluster to one hundred 200-byte records per cluster. The medium clusters collectively contain 8,400 medium records. Lastly, 140 small clusters range from forty 100-byte records per cluster to two hundred 100-byte records per cluster. The small clusters contain, collectively, 16,800 small records.

Database sizes. The size of the database is the sum of all the records in all the clusters (4×1.68 Mbytes = 6.72 Mbytes). For the RTR study, this figure must be a multiple of the factor 840. Since 6.72 Mbytes = $840 \times 8,000$ bytes, we can redistribute the same 6.72-Mbyte database evenly over two-, three-, four-, five-, six-, seven-, or eight-back-end configurations.

For the RTI study, we simply replicate the 6.72-Mbyte database n times for any n -back-end configuration. Thus, for the eight-back-end configuration, the benchmark database is 53.76 Mbytes (6.72 Mbytes $\times 8$).

Benchmark results. We performed separate RTR studies on all eight configurations by redistributing the same database for the five data-intensive and two overhead-intensive transactions, (minimally, eight database loads for 56 transaction runs). We summarize the results in Figure 6 (p. 56). Since there are seven benchmark transactions, there are seven RTR graphs in Figure 6. We have also completed separate RTI studies on eight configurations by replicating the database seven times for the five data-intensive and two overhead-intensive transactions (minimally, another eight database loads and another 56 transaction runs). Figure 7 on p. 57 shows the RTI statistics and graphs.

RTR results on data-intensive operations. Figure 6a shows the graph for Transaction 2, which accesses 96 percent of the

30,240 records in the 6.72-Mbyte database and selects 26 percent (7,560 records of various sizes) of the records retrieved as its output. It takes about 12 seconds to complete the RETRIEVE operation in the baseline configuration. As the same database is distributed to the two-back-end configuration evenly, we expect the ideal response time for Transaction 2 to be about 6 seconds. Since we doubled the back-end number, we expect the response time to be cut in half. Instead, the measured response time is about 7 seconds. Thus, the overhead incurred with two parallel back ends is about 1 second.

As we increase the back-end number and redistribute the same database, correspondingly, at one third each, one fourth each, one fifth each, one sixth each, one seventh each, and one eighth each for each new configuration, the measured performance curve continues to slide in Figure 6a. This indicates that the additional back ends cut down response time. Furthermore, the differences of the measured response time and the corresponding ideal response time for all configurations remain the same, about 1 second. In other words, this kind of parallelism does not increase the overhead.

In Figure 6b for Transaction 3, Figure 6c for Transaction 4, Figure 6d for Transaction 5, and Figure 6e for Transaction 7, the plotted curves all slide downward. The plotted curves again closely parallel the ideal curves, regardless of the number of back ends configured. These figures indicate that use of a multiplicity of back ends to reduce the response time of data-intensive operations is a promising approach. The curve has not yet leveled off at the eight parallel database processor-store pairs. We may approach a higher degree of parallelism beyond eight for more reductions in response time.

RTR results on overhead-intensive operations. In Figures 6f and 6g, we show the measured RTR statistics on Transactions 1 and 6. As overhead-intensive operations, these two transactions access only a small amount (248.8 Kbytes) of records data (4 percent of the 6.72-Mbyte database) from the database stores. However, they access a relatively large amount of in-memory processing (50 percent of the records retrieved for qualification or deletion). As we can see from the graphs, the curves level off at the five-back-end configuration. Thus, for overhead-intensive operations, there is no appreciable reduction in response time beyond this degree of parallelism. In other words, overhead-intensive operations do not require a highly parallel database computer. Instead, we should improve the performance of individual back ends with types such as RISC processors.

RTI data-intensive operations. In Figure 7a, we see that all five data-intensive operations, Transactions 2, 3, 4, 5, and 7, exhibit relatively level curves in all eight configurations, that is, from the baseline to the eight-back-end configuration. This indicates the response time of each transaction remains unchanged, or invariant, despite a change in database size from the baseline 6.72 Mbytes to 53.76 Mbytes in the eight-back-end configuration. In other words, even though

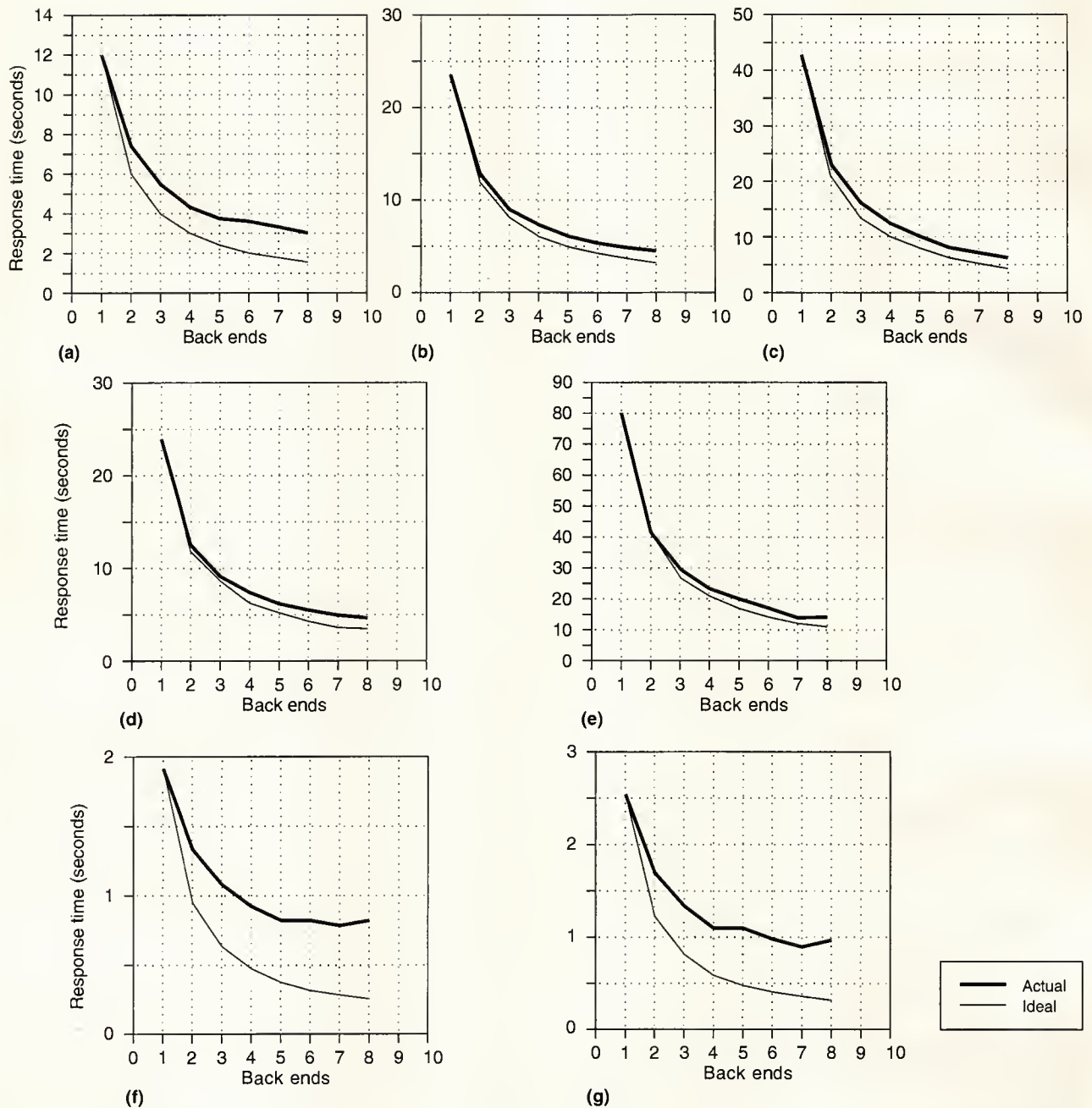


Figure 6. Response-time reductions over the multiplicity of parallel back ends for transactions 2 (a), 3 (b), 4 (c), 5 (d), 7 (e), 1 (f), and 6 (g).

the transaction must operate on more data, the response time of the transaction remains invariant, as long as the multiplicity of the parallel back ends is proportional to the increase in database size.

Transaction 7 (a DELETE operation) almost doubles the response time of Transaction 4 in every configuration. This is not surprising. Both access the same amount of data (the entire database). However, Transaction 7 must not only

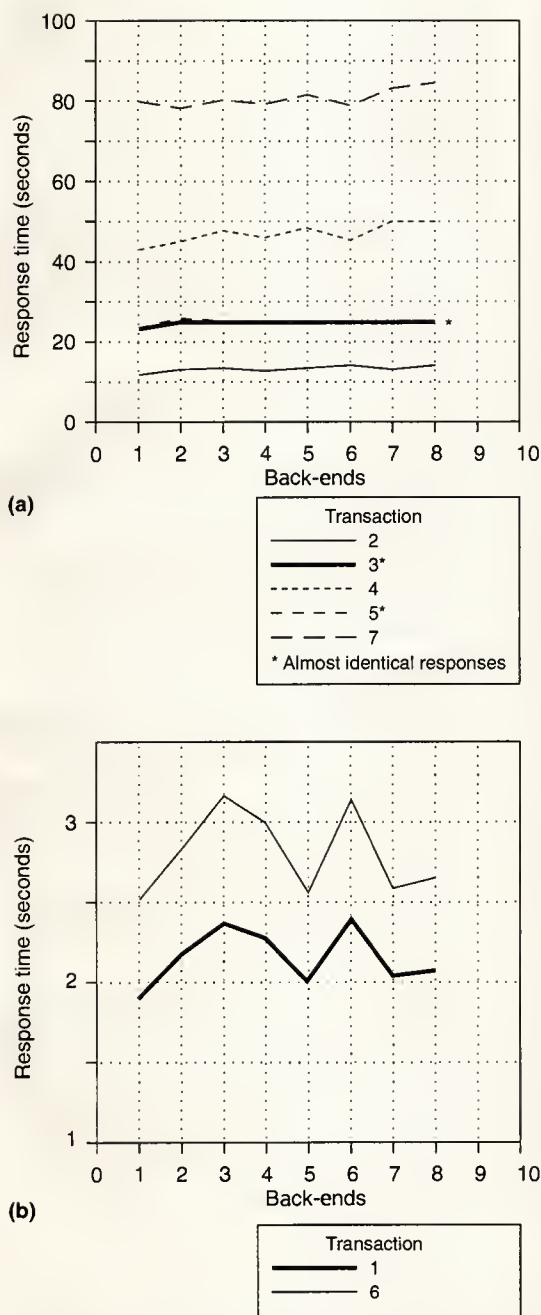


Figure 7. Response-time invariances over the multiplicity of parallel back ends for data intensive transactions (a) and for overhead-intensive transactions (b).

select 50 percent of the records accessed as does Transaction 4 (a RETRIEVE operation) but must also delete the selected records from the database stores. The curves indicate that it

takes some 40 seconds for each back end to access and process 30,240 records of varying sizes, and it takes some 30 seconds more for each back end to tag 15,120 records for deletion and return them to their database stores for later removal.

Transaction 5 (another DELETE operation) and Transaction 3 (a RETRIEVE operation) have almost identical response times for all configurations. This is also not surprising, since they both access the same half of the database and select half of the records retrieved (for deletion in the case of Transaction 5, or for returning data in the case of Transaction 3). But, when the amount of accessing is relatively small and the amount of processing is again relatively small, the times for writing the smaller number of deletion tags onto database stores are mostly overlapped by the accessing and processing times of the other records.

RTI results on overhead-intensive operations. Figure 7b contains the response times for the overhead-intensive operations in Transactions 1 and 6. Both curves exhibit a similar zigzag pattern. This indicates that, as the benchmark database is doubled, tripled, quadrupled, quintupled, sextupled, septupled, and octupled for the different configurations, the data has not been evenly redistributed on the corresponding configurations. The back end that has the highest number of records in a multiback-end configuration has the longest response time. Nevertheless, the deviation is within only one-half second in Transaction 1, and within 1 second in Transaction 6. It is important that, for either transaction, the response times deviate upward and downward from the norm. The norm for Transaction 1 is about 2.25 seconds. The norm for Transaction 6 is about 2.75 seconds.

MDBS limitations

There are several hardware and software limitations, but the prospects of MDBS-like database computers will be great if we can overcome the limitations.

Hardware. The hardware limitations seem to be related to the microprocessor-based Ethernet. A dual-ported, 124-Kbyte real memory supports two microprocessors in a back-plane. This real memory also buffers the messages coming from and going to the Ethernet cable. Since messages are typically small and infrequent, the real memory size is adequate for buffering messages. Further, messages are sent mostly via the point-to-point communication protocol, which cannot cause a collision.

For buffering records, though, the size of the real memory is inadequate, since records are individually large, typically hundreds of bytes per record, and are transmitted in bulk, typically tens, if not hundreds, of records in a batch transmission. The buffer, therefore, overflows, and some of the records are overwritten by records arriving later. This requires the sender to repeat the send protocol containing the last batch of records. Repetitions of this type markedly slow the broad-

cast operations in the second stage of a RETRIEVE COMMON operation where each back end broadcasts its subset of the second set of records to all the other back ends. As one or more back ends must repeat the broadcast operations, the delay and traffic become intolerable. This, of course, severely slows RETRIEVE COMMON.

Larger buffer memories and interrupt mechanisms. We make two recommendations for the purpose of overcoming this hardware limitation: provide a larger, dual-ported buffer memory in each backplane and provide an interrupt mechanism. Whenever the buffer memory is full, the backplane microprocessors interrupt the CPU (in our case, the back-end computer's 68020 microprocessor) so the CPU can begin the Get process to receive the records in the buffer right away. Without this interrupt hardware, there is no quick way to alert the back-end computer's CPU when the buffer is full and ready to forward the records. Meanwhile, Get, the sole process for receiving records or messages from the Ethernet for other processes, may be put on standby, while the CPU executes another process.

Higher bandwidth communication network. The Ethernet has a bandwidth of 10 Mbps. This is adequate for all the database primary operations, except RETRIEVE COMMON, where the bandwidth must sustain the cardinality of the second set of records in the broadcast mode. Say the cardinality is 25,000 records, and the average record size is 750 bytes. Then eight subsets of the second set of records of 18.75 Mbytes are broadcast at about the same time over the Ethernet. Not counting the delay due to eightfold collisions during the period of multibroadcasting, the present bandwidth requires at least 15 seconds to complete the transmission. Assuming that it takes 2 seconds to resend one eighth of the record set when a collision occurs, a minimum of 16 seconds are needed for the second set. This is also too slow.

Software limitations. Software limitations are related mainly to INSERT, UPDATE, and RETRIEVE COMMON, three of the five database primary operations.

Massive loading of a new database. For our benchmark effort, we created benchmark databases. Each database is a maximum 6.72 Mbytes per back end. We used INSERT to create each database by adding the records one at a time. This process is, of course, very time-consuming. It took us days to create one database and weeks to create all the necessary databases. We needed a massive loading program that could read the attribute-value pairs generated from a record generator, format them into records, extract their typed attributes, form their attribute table, and construct the descriptors from the typed attributes. We also needed it to form the descriptor table, determine the clusters of all records, form their cluster table, load the attribute, descriptor, and cluster tables onto the metadata disks, and load clustered records evenly onto base data disks one cluster of records at a time.

Control of virtual-memory space. In both UPDATE and

RETRIEVE COMMON, each back end must first select a set of records to be processed. Regardless of its size, this record set arrives from the database stores and temporarily resides in the virtual memory, awaiting subsequent operations. Typically, we use hashing techniques to quickly come up with addresses for such temporary storage. Unfortunately, the addresses are virtual—not real—addresses. Consequently, some or nearly all of the records in the temporary storage page out of real memory. It takes many paging (I/O) operations to retrieve the records for the next processing step in real memory. This delays or slows down both the UPDATE and RETRIEVE COMMON operations.

We make the following recommendations here for the purpose of overcoming this limitation:

- The operating system provides us with a system function that enables us to lock a certain number of virtual memory pages into real memory. So, as long as we use these locked pages for the temporary storage of our records, there will be no paging operations involved.
- On the basis of the size of available locked pages, we can partition either an UPDATE or a RETRIEVE COMMON into a series of updates (in which each update works on a new batch of records in the locked pages) or a series of RETRIEVE COMMONS. For example, in the first set of records each RETRIEVE COMMON works on a new batch of records in the locked pages. In other words, each batch of the first record set is processed against the entire second record set, although the second record set can also be brought in for processing one batch at a time (as long as both batches can be accommodated in the available locked pages).

System process priority. The Get process is a very high-priority process that should be constantly listening over the communication net and anticipating messages, data, or transactions heading its way. It is akin to a real-time process. In other words, the CPU of the back end should never deactivate a Get. The operating system should provide a system function that enables us to set the priority of one of our processes with the capability of real time.

DESPITE THE LIMITATIONS the prospects for a microprocessor-based, multiback-end database computer, such as MDBS, are good. These prospects are rooted in the following categories: external technology and intrinsic architecture.

We note several technology prospects.

- 1) All the hardware and software limitations discussed previously can be overcome with present computer technology. We do not have to wait for any distant technology for solutions.

Project support

The work reported here is now supported by funds from the Naval Research Laboratory and the Naval Postgraduate School. It began in 1980 with a proposal for an equipment grant submitted by the author to Digital Equipment Corp. In 1981 DEC and the Office of Naval Research funded the initial MDBS with a VAX 11/780 as the controller computer, two PDP 11/44s and their disk drives as back ends, and some PCLs as the communication network. The use of the 11/780 for the development of MDBS software was the main reason that we had also employed the VAX unit as the controller. This was before the advent of the microprocessor and the arrival of the microprocessor-based Ethernet. All three of our computers were minicomputers, with the VAX using the VMS operating system, and the PDP, the RSX operating system. The Ethernet functional specification had just been announced. We used three PCL cables to "emulate" the Ethernet specification. The two-back-end database computer was operated in the Laboratory for Database Systems Research at Ohio State University.

The MDBS software requirements were largely the work of J. Menon,¹ D. Kerr, A. Orooji, and S. Demurjian accomplished the principal supervision and development.²⁻⁴

The process structure is essentially the same to this day, although we have updated the original code many times. We wrote all processes in C, with five of them running in VMS and the other five in RSX. Get and Put in VMS communicate with Put and Get in RSX. The Office of Naval Research supported our software development effort as well.

In 1983, the Laboratory for Database Systems Research moved to the Computer Science Department of the Naval Postgraduate School. We gave MDBS's VAX 11/780 to Ohio State University and replaced it with one of the two Computer Science Department's VAX 11/780s. In 1985, we began to phase out and replace all the minicomputers and their communications network with microprocessor-based workstations and the Ethernet. We also began to add as many back ends as funds permitted. After we reached the round number of eight, we began to upgrade the microprocessors from 68010 to 68020 chips, and the real memories in both size and speed. The present configuration⁵ is depicted in Figure 1. We recently acquired two Sun-4 workstations with RISC-based microprocessors. However, we have not decided whether we should replace all the eight back ends with Sun-4s or simply add these two to the existing configuration for a heterogeneous, 10 back-end configuration. The replacement and upgrade were funded by the Naval Postgraduate School equipment program and the Department of Defense STARS (Software Technology for Adaptable, Reliable Systems) program.

MDBS at one time was known as MBDS. However, because of a trademark conflict, we changed the name of our database computer to its present name. MDBS is used as a research vehicle for the study of design analyses⁶⁻⁸ and performance evaluations⁹⁻¹¹ and their methodologies.¹² We also use it to study the database system architectures of the future.¹³⁻¹⁷

- 2) All the back-end hardware requirements for a database computer require no special-purpose computer components or devices. They can be met with off-the-shelf microprocessor-based hardware, such as Sun-4 workstations or super PCs, each of which can support several hard disks of various capacities.
- 3) The LAN hardware requirements for a database computer can also be met with an Ethernet-like LAN with wider bandwidths, higher transmission rates, and reliable multibroadcast capabilities. Such LANs can be made available commercially.
- 4) Microprocessor technology is progressing at a higher rate than that of both minicomputers and mainframes. We should be able to ride on the success of these faster advances.
- 5) In storage technology, Winchester-type disks and disk arrays based on that principle are improving more rapidly than are the densities and capacities of standard disk drives. By relying on the gains in Winchester-type

drive technology, we should have a better mix and choice of disk drives with different capacities and numbers for separate metadata, base data, and page stores.

We also note some architectural prospects.

- 1) Parallel architecture uses identical back ends. Also, database back-end software and metadata are replicated. Only base data require redistribution. Thus, the cost of any additional hardware and software is minimized. It is rather straightforward to increase parallelism readily and cost-effectively.
- 2) An increase in parallelism can be scaled for the purpose of achieving the desired response time for a given transaction and/or a certain capacity of a database.
- 3) The increase in performance for a given response time reduction or a capacity increase for a response time invariance is directly proportional to the number of back ends and corresponding stores added.

The MDBS architecture is the most effective and efficient one in reducing the response time of a transaction. It is also the most effective and efficient architecture to maintain the same response time of a transaction, despite the increase in size of a database. ■

Acknowledgments

The MDBS software development involved a large number of undergraduates, graduates, postgraduates, professors, and other professionals. They cannot all be named here without committing omissions. However, D. Kerr, A. Orooji, and S. Demurjian accomplished the principal supervision and development.

References

1. S.A. Demurjian, D.K. Hsiao, and J. Menon, "A Multiback-End Database System for Performance Gains, Capacity Growth, and Hardware Update," *Proc. Second Int'l Conf. Data Engineering*, IEEE CS Press, Los Alamitos, Calif., 1986.
2. D.K. Hsiao et al., "The Implementation of a Multiback-end Database System (MDBS): Part I, An Exercise in Database Software Engineering," *Advanced Database Machine Architecture*, Ch. 10, Prentice-Hall, Englewood Cliffs, N.J., 1983.
3. H. He et al., "The Implementation of a Multiback-end Database System (MDBS): Part II, The Design of a Prototype MDBS," *Advanced Database Machine Architecture*, Prentice-Hall, 1983, Ch. 12.
4. R.D. Boyne et al., "A Message-Oriented Implementation of a Multiback-end Database System (MDBS)," *Database Machines*, H. Leilich and M. Missikoff, eds., Springer-Verlag, New York, 1983.
5. D.K. Hsiao, "Super Database Computers: Hardware and Software Solutions for Efficient Processing of Very Large Databases," *Proc. IFIP 86 Congress*, IFIP Press, Geneva, 1989.
6. D.K. Hsiao, "Cost-Effective Ways of Improving Database Computer Performance," *AFIPS Conf. Proc.*, Vol. 52, AFIPS Press, Reston, Va., 1983.
7. D.K. Hsiao and P. Strawser, "The Predicate Machine—A High-Level Database Computer," *Proc. Int'l Conf. High-Level Language Computer Architecture*, 1984.
8. D.K. Hsiao, "The Impact of the Interconnecting Network on Parallel Database Computers," *Database Machines*, H. Tanaka and M. Kisuregawa, eds., Kluwer Academic, Boston, 1987.
9. S.A. Demurjian et al., "Performance Evaluation of a Database System in Multiback-end Configurations," *Database Machines*, D.J. DeWitt and H. Borel, eds., Springer-Verlag, 1985.
10. S.A. Demurjian et al., "A Computer-Aided Benchmarking System for Parallel and Expandable Database Computers," *Proc. 1987 Fall Joint Computer Conf.*, AFIPS Press, 1987.
11. J.E. Hall, D.K. Hsiao, and M. Kamel, "Performance Evaluations of a Parallel and Scalable Database Computer," *Proc. Conf. Databases, Parallel Architectures, and Their Applications*, N. Rishe, ed., IEEE CS Press, 1990.
12. S.A. Demurjian, D.K. Hsiao, and R. Marshall, *Design Analysis and Performance Evaluation Methodologies for Database Computers*, Prentice-Hall, 1987.
13. D.K. Hsiao, "Future Database Machine Architectures," *New Directions for Database Systems*, Ch. 2, R. Ariav and A. Clifford, eds, Ablex Publishing Corp., Norwood, N.J., 1986.
14. S.A. Demurjian and D.K. Hsiao, "Towards a Better Understanding of Data Models through the Multilingual Database System," *IEEE Trans. on Software Engineering*, Vol. 14, No. 7, July 1988, pp. 946-950.
15. D.K. Hsiao and M. Kamel, "Heterogeneous Databases: Proliferations, Issues, and Solutions," *IEEE Trans. Knowledge and Data Engineering*, Vol. 1, No. 1, Mar. 1989, pp. 45-62.
16. D.K. Hsiao, "Databases and Database Systems in the 21st Century," *Very Large Scale Computation in the 21st Century*, Ch. 9, J. Mesirov, ed., SIAM, Philadelphia, 1991.
17. D.K. Hsiao, "Federated Databases and Systems, A Tutorial," (to appear in) *Int'l J. Very Large Data Bases*, Vol. 1, No. 1, Mar. 1992.



David K. Hsiao is a professor of computer science at the Naval Postgraduate School in Monterey, California. His technical interests include database systems and computers, database security and access control, and federated heterogeneous databases and systems.

Hsiao earned BS and MA degrees in mathematics from Miami University and a PhD in computer and information science from the University of Pennsylvania. He became an IEEE Fellow in 1989 and also served as a member of the governing board of the IEEE Computer Society. He was an elected member-at-large of the ACM Council and is a trustee of its Very Large Data Base Endowment.

Address questions concerning this article to David K. Hsiao, Computer Science Department, Naval Postgraduate School, Monterey, CA 93943; or via e-mail at hsiao@cs.nps.navy.mil.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162	Medium 163	High 164
---------	------------	----------



Rinda:

A Relational Database Processor with Hardware Specialized for Searching and Sorting

This database processor accelerates nonindexed relational database queries. Rinda is composed of content search processors and relational operation accelerating processors; the former search rows stored in disk storage, and the latter sort rows stored in the main memory. The processors connect to a general-purpose host computer with channel interfaces. Performance improves substantially as Rinda executes queries three to 100 times faster than conventional database-management system software in a benchmark system.

Ushio Inoue

Tetsuji Satoh

Haruo Hayami

Hideaki Takeda

Toshio Nakamura

Hideki Fukuoka

NTT Communications

and Information Processing

Laboratories

Various database machine architectures have been proposed and developed to solve the performance problems of relational databases. Database machines fall into two classes on the basis of their functions: computers and processors. Database computers, such as the Britton Lee Intelligent Database Machine and the Teradata DBC/1012, are independent of their host computers and perform all database-management functions by themselves. Database processors, on the other hand, depend on their hosts and perform only certain database functions. For example, CAFS¹ is dedicated to ICL mainframes, and its functions are limited to searching and filtering table rows stored in disk storage. IDP² is an optional processor for Hitachi's largest mainframe, and it can perform only sorting and merging of keys stored in the main memory. In general, database processors achieve higher performance at lower costs, because specialization in their hardware is restricted to database functions that have caused severe performance problems in general-purpose computers.

We can classify relational database access into queries and updates, which we can further classify into nonindexed and indexed queries and updates.³ Database systems execute indexed queries and updates efficiently using indexes created

before the execution. A typical indexed query is a selection of a single row in a table by exact matching of a unique search key. With such queries or updates, current database-management systems perform fairly well. For nonindexed queries, however, systems cannot take advantage of indexes. Examples of nonindexed queries are a selection of multiple rows by partial matching of a character string, a join of two tables with nonunique join keys, and an aggregation of a whole table with groups of rows in it. A general-purpose computer consumes a lot of processing time executing such queries because searching and sorting are computationally intensive operations.⁴

At NTT, we developed a database processor called Rinda to accelerate nonindexed relational database queries.³ Rinda is composed of content search processors and relational operation accelerating processors.⁵ A content search processor, or CSP, searches rows in disk storage and transfers the selected rows to the main memory. A relational operation accelerating processor, or ROP, sorts the rows stored in the main memory. The CSP and ROP perform their work using hardware specialized for searching and sorting. Rinda is connected to a general-purpose host—our DIPS series mainframe^{6,7}—with channel interfaces, and is controlled by a Rinda control program running on the host.

Rinda architecture

In relational database systems, searching and sorting are basic operations. Searches select rows and columns from a table according to predicates in a query and are the first step in query execution. When a table is stored on disk, disk reads are also needed. There are two problems with searches. For nonindexed queries, they take a great amount of CPU time because all rows in a table must be qualified. Transferring all rows from the disk to the main memory also takes extra time. A solution to these problems is an intelligent disk controller that performs searches at the disk storage.

Sorts order rows in a table using a key, which may be composed of several columns. Systems can perform join, nested, and aggregate queries efficiently when rows are sorted. The major problem in sorting is CPU time consumption, because its complexity with ordinary algorithms is $N * \log(N)$, where N is the number of rows. Another problem stems from memory storage because additional disk accesses are required when memory size is insufficient. A solution to these problems is a hardware sorter with a large memory, which sorts rows in time N .

Design considerations. Our goal in developing Rinda was to relieve host computers from heavy loads caused by nonindexed queries. Therefore, Rinda had to satisfy the following requirements:

- 1) It should be a database processor, so the existing host computer and disk storage would not need to be replaced.
- 2) The application program interface should be SQL to avoid any modification of user programs.

To meet the first requirement, Rinda uses the two special-purpose processors just introduced: CSPs and ROPs. A CSP connects to a host computer and a disk controller with channel interfaces, forming an intelligent disk controller. An ROP that uses a hardware sorter also connects to the host by a channel. To meet the second requirement, we based Rinda's functions on a subset of SQL functions. Software on the host computer fills the gaps between full-set SQL and Rinda.

Hardware architecture. Figure 1 shows a typical Rinda system organization. The major components are a DIPS series host computer, standard disk controllers and disk drives, CSPs, and ROPs. Database size and performance requirements determine the number of CSPs and ROPs in a system. For example, if the database is very large, several CSPs used

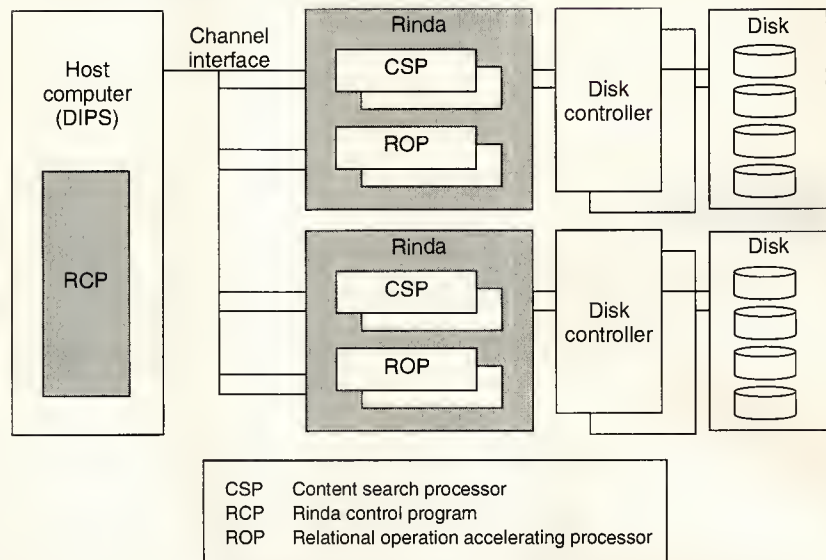


Figure 1. Rinda system organization.

Table 1. Primary CSP functions.

Function	Description
Predicates	Specified in a WHERE clause
Comparison	<column> <comp-op> <value>
In	<column> [NOT] IN <value list>
Like	<column> [NOT] LIKE <pattern>
Null	<column> IS [NOT] NULL
Boolean expression	Any combination of predicates
Column selection	SELECT <column list>
Set function	Count(*)

in parallel reduce the I/O time. Channel commands and order tables created by the Rinda control program, or RCP, and running on the host control the CSPs and ROPs.

A CSP searches a database table stored on a disk, selects rows and columns specified by a query and transfers only the results to the host. Table 1 shows its primary functions, which cover most single-table queries of relational databases. The CSP performs these functions at the disk's data-transfer rate.

An ROP sorts rows in a table transferred from the main memory of the host and transfers the results back to the host. Its primary functions, listed in Table 2, include removal of unnecessary rows to accelerate join and nested queries. The ROP performs these functions at the data-transfer rate of the channel. We discuss CSP and ROP hardware structure in detail later.

Table 2. Primary ROP functions.

Function	Description
Sorting	Order By <column[ASC/DESC] list> (also used for joins, subqueries, and Group-By clauses)
Filtering	Removal of unnecessary rows (used for join and nested queries)
Duplicate removal Set function	DISTINCT <column list> Count(*) with a Group-By clause

Software architecture. Figure 2 shows the host computer's software structure. The Rinda control program is attached to an existing relational database-management system to form a new integrated DBMS. The language-processing subsystem analyzes queries written as SQL statements. If the query is nonindexed, the system calls RCP functions, which optimize and execute queries using Rinda.

Figure 3 illustrates a typical procedure to execute a nonindexed query. First, the RCP sends a command to the CSPs, which transfer selected rows to the RCP buffers in the main memory. Next, the RCP sends another command to an ROP, which sorts the rows in the buffers. Finally, the RCP returns the results to the user program.

The RCP also allows a table to be stored over multiple disks and conducts parallel CSP searching on the disks. The database control subsystem controls RCP query execution to keep the database consistent when nonindexed queries and indexed updates execute concurrently in multiuser environments.

Related work. Several searching and sorting processors have been developed. For example, CAFS¹ is a searching processor with functions similar to those of the CSP. However, the CSP has properties different from those of other searching processors. First, its design is based on SQL functions, and it supports the null value. Second, it deals with the page format of an existing DBMS so no data conversion is required to introduce Rinda to installed user database systems. As for sorting processors, existing hardware sorters⁸ can sort only a small number of rows at one time because the number of processing elements is limited. The ROP solves this problem with multiway merging. Moreover, we integrated sorting and

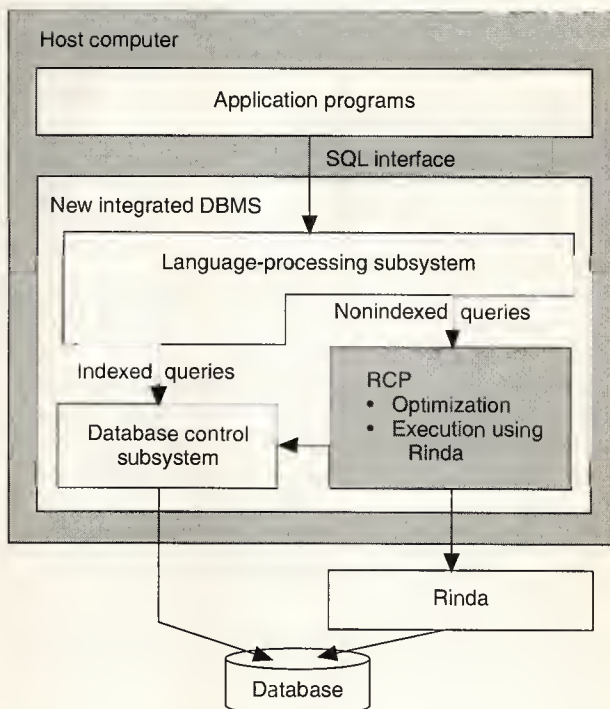


Figure 2. Software structure on a host computer.

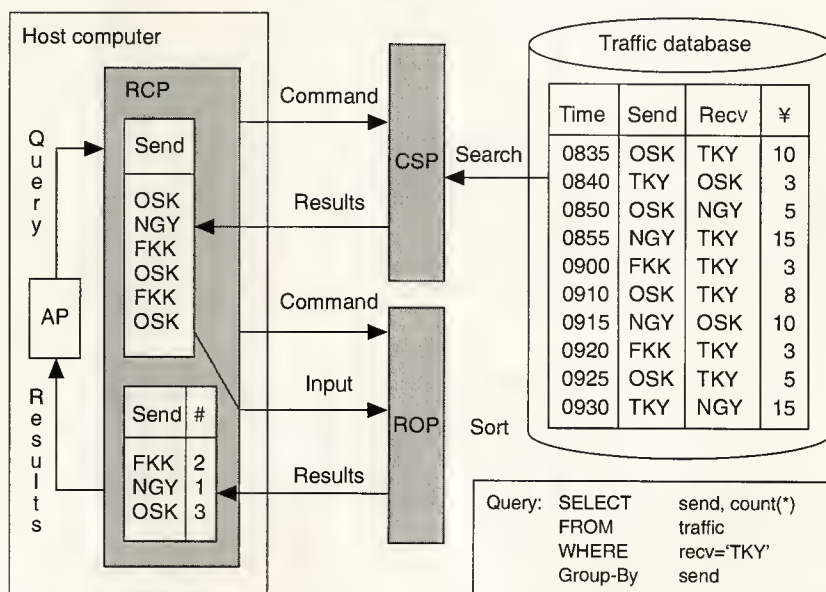


Figure 3. Typical query execution using Rinda (AP indicates the application program).

filtering in the ROP to use the three-phase join method.

Content search processor

A database consists of database spaces, each of which is a collection of pages stored in one or more continuous disk areas. A database space may be distributed over several disks. When a table is created in a database space, adjacent pages are assigned to the table. As the table grows, a constant addition of adjacent pages increases the table. Thus, a table is stored in multiple disk extents. Each page holds rows of a single table, and no row is stored across page boundaries. Each table contains control records indicating its disk extents in separate pages, and the RCP uses this information to perform CSP searches.

CSP organization. A CSP search can be performed in two ways, synchronously and asynchronously. In the synchronous mode, a data stream from a disk processes on the fly, while in the asynchronous mode, buffers decouple reading and searching. The CSP performs only asynchronous searches for the following reasons:

- A disk controller has an error-detecting and -correcting facility for data pages. This facility completes after the last byte of the page transfers from the disk to the buffer.
- We had designed the page format without any consideration of hardware-searching mechanisms. Changing the page format from software-oriented to hardware-oriented was impossible because it would require database conversion for existing systems.

The CSP transfers pages successively from a disk to its buffers using the multitrack-read method and scans the page in the buffer before the next page comes. Therefore, the CSP performs a search within the page-transfer time, similar to doing it on the fly. Figure 4 shows a block diagram of the CSP.

Search operation flow. Rinda systems perform searches as follows:

- 1) The RCP running on the host acquires the RCP buffers and prepares a set of channel commands and an order table for each CSP. An order table contains disk access information that includes the disk unit number, extent addresses to be searched, the RCP buffer size, and a search condition.
- 2) Channel commands send the order tables from the host to the CSPs.
- 3) According to the order table, each CSP generates new channel commands to the disk to read pages successively using the multitrack-read method. When the first page arrives, the CSP begins to select rows from the page. To

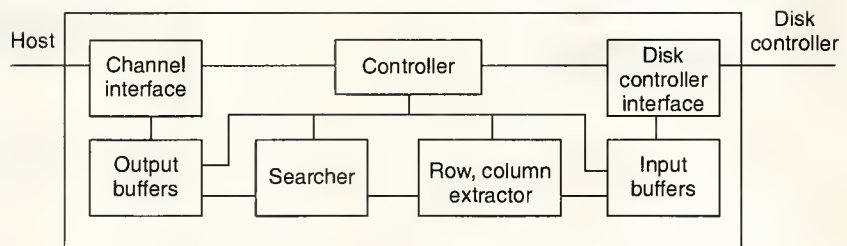


Figure 4. CSP block diagram.

keep up the data-transfer rate, the CSP uses multiple input buffers that hold pages transferred from the disk, and multiple output buffers that hold new pages containing selected rows.

- 4) When one of the output buffers fills, the CSP transfers the page to the host asynchronously.

This procedure continues without any interruption to the host until the CSP has read all the extents or until all the RCP buffers are occupied. In the latter case, the RCP saves the selected rows on a work disk and repeats the search procedure.

Evaluation of search conditions. A search condition is expressed by predicates and Boolean operators And, Or, and Not. In SQL, when the value is undefined, each column may have a null value instead of zero or space. As a result, the truth value of a predicate in the condition may be true, false, or unknown. Therefore, the three-valued logic (true, false, and unknown) shown in Figure 5, instead of two-valued logic (true and false), defines Boolean operators.

And	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

Or	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

Not	True	False	Unknown
	False	True	Unknown

Figure 5. Truth tables of the three-valued logic.



December 1991 issue (card void after June 1992)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest

(Add comments on the back)

Readers, indicate your interest in articles and departments by circling the appropriate number (shown on the last page of articles and departments):

150 151 152 180 181 182
153 154 155 183 184 185
156 157 158 186 187 188

159 160 161 189 190 191
162 163 164 192 193 194
165 166 167 195 196 197

168 169 170 198 199 200
171 172 173 201 202 203
174 175 176 204 205 206

177 178 179 207 208 209

Product Information 1

(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	



December 1991 issue (card void after June 1992)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest

(Add comments on the back)

Readers, indicate your interest in articles and departments by circling the appropriate number (shown on the last page of articles and departments):

150 151 152 180 181 182
153 154 155 183 184 185
156 157 158 186 187 188

159 160 161 189 190 191
162 163 164 192 193 194
165 166 167 195 196 197

168 169 170 198 199 200
171 172 173 201 202 203
174 175 176 204 205 206

177 178 179 207 208 209

Product Information 2

(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

☐ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only \$21 for a year's subscription (six issues).

Society: _____

IEEE membership no: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate (\$10.50) for three bimonthly issues.

Full Signature _____ Date _____

Name _____

Street _____

City _____

State/Country _____ ZIP/Postal Code _____

☐ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society, IECEI, IPSI, NSPE, SCS, or other professional society, pay the sister-society rate of only \$38 for a year's subscription (six issues).

Organization: _____

Membership no: _____

☐ Payment enclosed Residents of CA, DC, Canada, and Belgium add applicable sales tax.

☐ Charge to ☐ Visa ☐ MasterCard ☐ American Express

Charge-card number _____

Expiration date _____

Month _____ Year _____

Prices valid through 12/31/91
12/91 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: _____

I disliked: _____

I would like to see: _____

Reviewers Needed. If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.

For reader-service inquiries, see other side.

PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

IEEE Micro

PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA



Editorial comments

I liked: _____

I disliked: _____

I would like to see: _____

Reviewers Needed. If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.

For reader-service inquiries, see other side.

PO Box is for reader-service cards only.

PLACE
POSTAGE
HERE

IEEE Micro

PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY

PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



If unknown is replaced by false in the evaluation of predicates, the three-valued logic is localized into two-valued logic, and the CSP implements simply. A where clause in an SQL statement, WHERE <search condition>, selects rows whose final truth value of the <search condition> is true. Therefore, unknown can be replaced by false in the last stage of the evaluation. In the truth table shown in Figure 5, the Not(unknown) column is important. If unknown is simply replaced by false, the result may be unreasonable because Not(false) is true while Not(unknown) is false. Therefore, if there is no Not operator in the search condition, unknown can be replaced by false, and the three-valued logic can be localized into two-valued logic.

The following procedure transforms any search condition to another search condition having the same effect without Not operators:

- 1) Transform a search condition to a conjunctive canonical form.
- 2) If there is no predicate with Not, the procedure ends.
- 3) If there is a predicate with Not, remove Not in the predicate and reverse the comparison operator in the predicate. For example, Not(column1 > x) is replaced by (column1 ≤ x).

In Rinda systems, the RCP transforms a search condition to a conjunctive canonical form without Not operators, and creates an order table for the CSP. The CSP replaces unknown with false in the evaluation of predicates and applies two-valued logic.

Accelerating processors

The purpose of the ROP is to accelerate join queries. We developed a new join algorithm, which makes the best of a hardware sorter.

Join algorithms. There are three principal types of conventional join algorithms: nested-loop,⁹ sort-merge,⁹ and hash¹⁰ algorithms. Nested-loop join algorithms repeatedly compare each row in the outer table with all rows in the inner table. This algorithm is practical only if both tables are small, because it requires a very large number of comparisons. Hash join algorithms split both source tables into several buckets using a hashing function and execute comparisons in each bucket when every row has the same hashing value. Therefore, hash join algorithms are suitable for parallel processors. Sort-merge join algorithms sort both tables to decrease the amount of merge-join computations to a linear order. We selected a sort-merge join algorithm for Rinda because specialized hardware can rapidly execute sorting.

Rinda's three-phase join method^{4,11} based on the sort-merge algorithm consists of filtering, sorting, and merge-join phases. Most unnecessary rows are removed in the filtering phase. Remaining rows, each of which is a candidate of the join, are

sorted in the sorting phase. After this, sorted rows are merged together in the merge-join phase.

The filtering phase uses hashed bit arrays¹² to remove unnecessary rows. Of the three phases, the filtering phase involves the most rows. Therefore, we decided that specialized hardware should execute this step. Sorting consumes much CPU time, so we chose specialized hardware to execute the sorting step. Filtering decreases the number of rows, and the remaining rows are sorted during the sorting phase. Therefore, the RCP on the host executes the merge-join phase with little computation.

ROP organization. Figure 6 shows a block diagram of the ROP, which performs filtering and sorting. Its main functional blocks are the hasher and sorter. The key extractor and row transfer blocks were implemented for a row-level pipeline.

Rows are generally composed of several columns with different data types such as integer, decimal, and character string. A null value may also appear in a column. The ROP has key-extractor hardware to compose a fixed-length and directly comparable internal key from a row. Using the internal keys, simple hardware can rapidly execute filtering and sorting.

The ROP has a single memory for storing internal keys, rows, and hashed bit arrays. The ROP moves the boundaries between these areas dynamically when it stores keys and rows. The constant ratio of the storage capacity determines the size of the bit array to keep the loading factor low. For example, if storage capacity increases, so does the size of the bit arrays. Keys and rows are stored in the remaining ROP memory.

Filtering block. Since unnecessary rows remain from collisions of hashing-function values, we needed a sophisticated hashing function to decrease the number of collisions. An ideal hashing function would distribute all keys uniformly over the addressing space of a hashed bit array. Lum, Yuen, and Dodd¹³ and Knott¹⁴ compared division, multiplication, folding, and other hashing functions on the basis of the number of collisions, using fixed-length short keys. The division

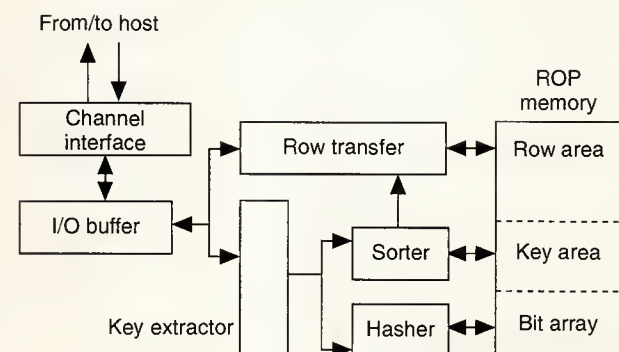


Figure 6. ROP block diagram.

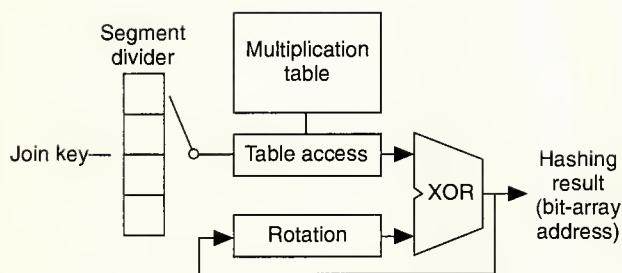


Figure 7. Multiplication-folding function.

method produced good results with fewer collisions for the unknown set of keys. However, in the filtering phase, the division method cannot be applied directly because internal keys may be too long. We determined that Rinda's hashing function should

- operate well when the loading factor, the number of hashing keys over the size of a bit array, is rather small;
- decrease the number of collisions (but it need not handle overflow when a collision occurs); and
- use the same hardware mechanism to hash any keys having various data types and lengths.

For hashing long keys, the folding method works well. It divides keys into several short fragments and folds them using the Exclusive-Or (XOR) operator. However, in character strings, especially in Japanese 16-bit kanji strings, the probability of 1 occurring in each bit is not even. Thus, hashed results have biases if the simple folding method with 1-byte or 2-byte fragments is used. Shifting or reversing the fragments in the bit order solves this problem.

We developed a new multiplication-folding function¹¹ for Rinda: a folding method with bit shifting and multiplication. The multiplication randomizes the character code, and the folding handles long keys. Compact specialized hardware with XOR and simple multiplication circuits implements the method.

Figure 7 is a schematic diagram of the multiplication-folding function. Keys are divided into several short segments. The first segment is multiplied by a prime number and rotated lap-around to distribute the value. Then, the second segment is multiplied by the same prime number and folded on the first segment with XOR. This procedure repeats until all segments are folded. This multiplication-folding function yields a high filtering factor for any

type and length of keys.

Sorting block. Many sorters can increase sorting speed using parallel and pipelined hardware.⁸ However, such hardware sorters are too large for database processors because their hardware size depends on the number of sorted keys. We decided that Rinda's sorter should

- handle a variable number of sorting keys at query execution time,
- achieve a high sorting speed regardless of the number of keys,
- handle any length and type of key efficiently because keys may be long and composed of several data types,
- use compact hardware, and
- permit easy expansion of the allowable number of keys.

The multiway merge sorter¹⁵ using a sorting array¹⁶ meets these requirements. As Figure 8 shows, the sorter implemented in the ROP consists of a sorting array, a merge controller, and working storage, which is the key area of ROP memory. The sorting array is composed of cascade-connected sorting elements with a one-dimensional linear array structure. Each element consists simply of two memories and a comparator as main circuits. The sorting array executes comparison-transferral actions synchronously in a pipelined, parallel manner. Each sorting element compares two keys and selects the smaller or larger one to the neighboring element through a dedicated data path controlled by the previous comparative result.

The sorter can compare a large number of keys in a series of k -way merge operations. In each k -way merging stage, k strings in the working storage are merged at once and then restored as a single string. The size of this string is the total number of keys in all k strings.

Using a new data-driven string-selection technique, the sorter performs a multiway merge operation. Keys, read from the working storage, are identified with bank tags. The sort-

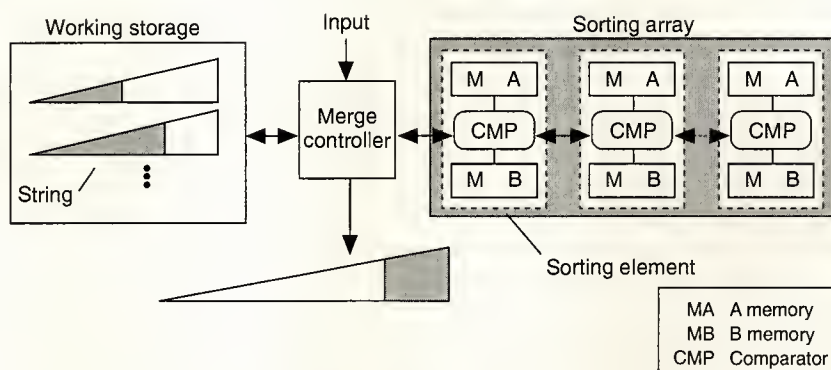


Figure 8. Multiway merge sorter.

Figure 9 shows a continuous key flow using the bank tag. The figure shows an example of four-way merging in descending order. In the M1 phase, the sorting array is filled with the largest keys in all source strings. These largest keys located at the top of the strings. In the M2 phase, merging proceeds successively. The largest key in the sorting array is immediately output with the bank tag. The second-largest candidate key is limited to any of the remaining keys in the sorting array or to the top key in the string indicated by the output key's bank tag. Therefore, this top key is input to the array and is compared with the remaining candidate keys. The sorter performs successive multiway merging using these alternate key output-input operations until all source strings are emptied.

We evaluated Rinda's performance using the extended Wisconsin Benchmark.^{17,18} The database table consisted of 100,000 rows, and each row was 208 bytes long. The Rinda system used the NTT DIPS-V30E superminicomputer as the host, and two CSPs and an ROP. The database resided on two 1.3-Gbyte disks whose data-transfer rate was 3 Mbytes per second.

Results. Figure 10 shows the performance improvement achieved with Rinda. The speedup is the elapsed time without Rinda divided by the elapsed time with Rinda. It is shown with the CPU and I/O times separated. The I/O time includes the processing time of the CSPs and the ROP and the time required to access the work disk. The system executes the queries for 1-percent and 10-percent selections and scalar Min and Count with only the CSPs but performs the others with a combination of the CSPs and the ROP. Table 4 lists the query execution time in seconds with Rinda as well as the

Considerations. Figure 10 shows that the speedup of queries with CSPs depends on the number of result rows. In general, the speedup is smaller as the results are larger. For example, the speedup is more than 50 times in the scalar Count in which no rows are transferred, while it is less than four times in the 10-percent selection. The reason is that the RCP requires more CPU time to transfer rows from the result table to the user program. In other words, the fetch operations take a long time when the number of result rows is large, because the host computer must perform them serially. The speedup of the scalar Min is less than that of the scalar Count, because the Count is directly supported by the CSPs while the Min is not. As for the Min, the specified column's values of all rows are transferred from the CSPs to

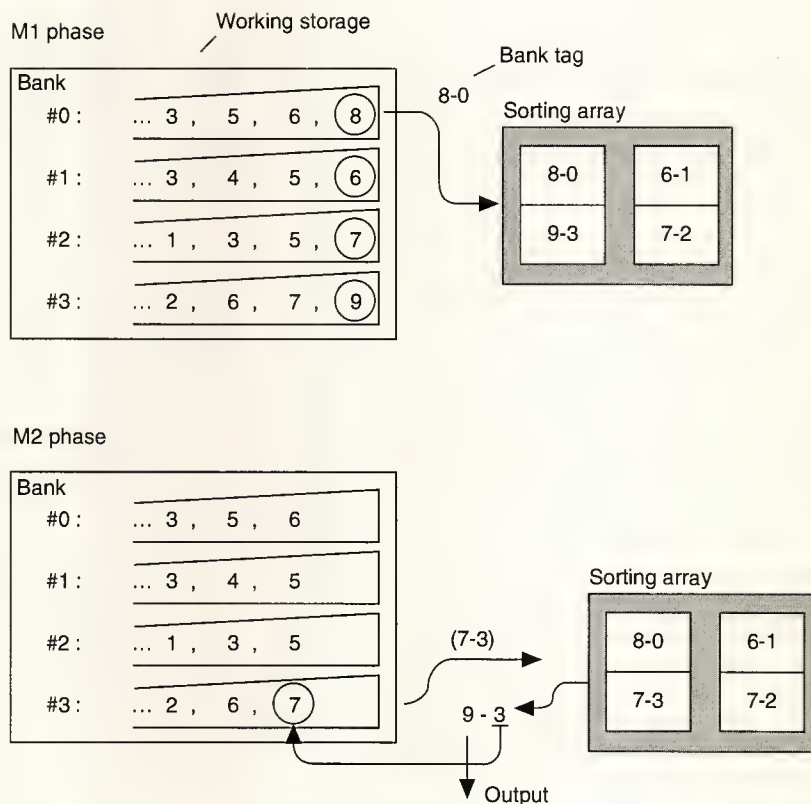


Figure 9. Continuous multiway merging using bank tags.

Table 3. SQL statements executed.

Query	Type	SQL statement
Selection	1 percent	SELECT * FROM HUNKA WHERE UNIQUE1>=50000 AND UNIQUE1<51000
	10 percent	SELECT * FROM HUNKA WHERE UNIQUE1>=50000 AND UNIQUE1<60000
Join	AselB	SELECT * FROM HUNKA A, HUNKB B WHERE A.UNIQUE1=B.UNIQUE1 AND A.UNIQUE1<10000
	CselAselB	SELECT * FROM HUNKA A, HUNKB B, TENKA C WHERE C.UNIQUE1=A.UNIQUE1 AND A.UNIQUE1=B.UNIQUE1 AND A.UNIQUE1<10000 AND B.UNIQUE1<10000
Min	Scalar	SELECT MIN(UNIQUE1) FROM HUNKA
	Group-By	SELECT MIN(TWOTHOU5) FROM HUNKA GROUP BY HUNDRED
Count	Scalar	SELECT COUNT(*) FROM HUNKA
	Group-By	SELECT COUNT(*) FROM HUNKA GROUP BY HUNDRED

the RCP, which determines the minimum value.

Comparison of the scalar and Group-By in the Min or Count functions in Figure 10 shows the speedup achieved with an ROP. The additional sorting load does not reduce the speedup, because the ROP sorts dozens of times faster than the host computer. The join queries also demonstrate the ROP's effect in contrast with the 10-percent selection, which does not use the ROP.

Table 4 shows that Rinda is faster in the selections and slower in the joins and Min functions than the Teradata and Gamma machines. The times differ because the host computer has almost nothing to do for the selections, while it must perform the merge-join phase for the joins and value comparisons for the Min function. These loads are somewhat smaller than those of the original queries. However, they still require relatively long processing times in single small-scale general-purpose processors.

Table 4. Query execution times in seconds.

Query	Type	Rinda	Teradata	Gamma
Selection	1 percent	5.2	18.6	13.4
	10 percent	9.0	14.9	12.7
Join	AselB	136.8	235.6	35.8
	CselAselB	128.5	95.7	37.9
MIN	Scalar	31.7	18.3	15.5
	Group-By	47.3	27.1	19.4

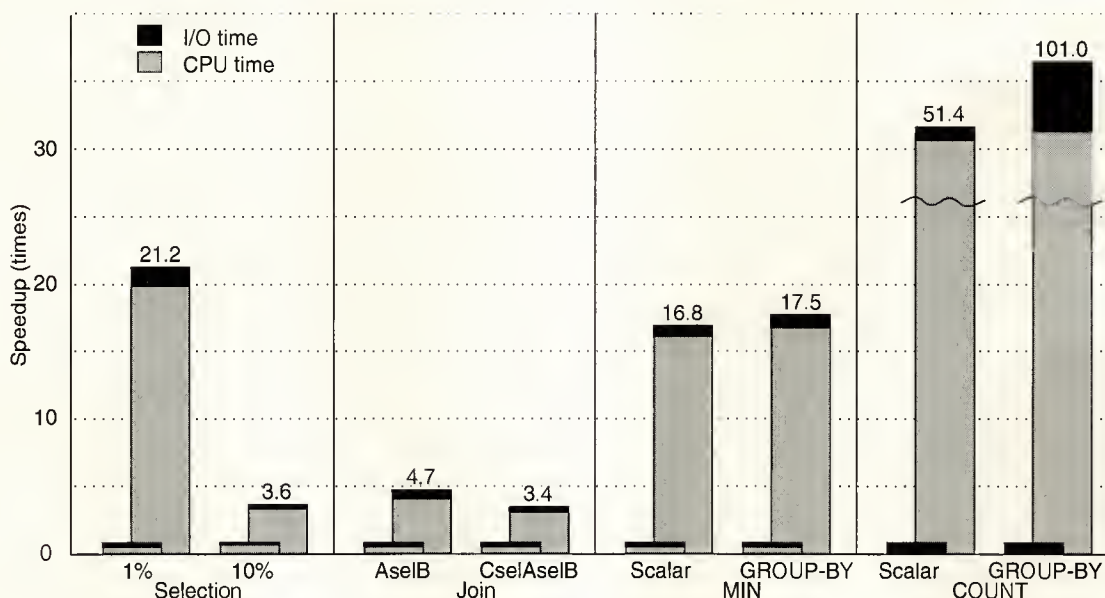


Figure 10. Speedup with Rinda (left) and without Rinda (right).

Merits and limitations

Rinda accelerates searching and sorting 20 to 50 times, and its total speedup of queries is in the range of three to 100 times. Rinda is generally very efficient when the source table is large and the result table is small.

However, Rinda has limitations, the first of which is concurrent execution with updates. To guarantee transaction serializability, the RCP (Rinda control program) must lock the whole table before CSP searching, resulting in a locking overhead with nonindexed queries and time delays with indexed updates. Using a dirty read facility that requires no locks for queries executed by Rinda solves this problem. The second limitation involves multiple query execution. The RCP executes multiple queries concurrently, while each CSP and ROP accepts only one request at a time. Therefore, simultaneous execution of two queries that require the same CSP or ROP increases response time.

RINDA RELIEVES ITS HOST COMPUTER OF THE HEAVY loads caused by nonindexed queries. The CSP, a special-purpose processor with hardware for searching, selects rows from a disk at the disk's data-transfer rate. The ROP, another processor with specialized hardware, sorts rows selected by the CSPs. For join queries, the ROP uses the three-phase join method with a hardware filter and sorter. Rinda substantially reduces the host computer's CPU and I/O times. Our performance study shows that Rinda accelerates nonindexed queries from three to 100 times, compared with conventional DBMS software.

Rinda operates with several business database systems. The businesses use Rinda mainly to retrieve rows by ad hoc queries or to get statistical reports from very large tables. Rinda's role is increasing as database applications become more advanced. ■

Acknowledgments

We thank Masato Haihara and Kenji Suzuki of NTT for their continuous support of this work.

References

1. E. Babb, "Implementing a Relational Database by Means of Specialized Hardware," *ACM Trans. Database Systems*, Vol. 4, No. 1, Mar. 1979, pp. 1-29.
2. K. Kojima, S. Torii, and S. Yoshizumi, "IDP—A Main Storage Based Vector Database Processor," *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic, Boston, 1988, pp. 47-60.
3. U. Inoue et al., "RINDA—A Relational Database Processor for Non-Indexed Queries," *Proc. First Int'l Symp. Database Systems for Advanced Applications*, Apr. 1989, pp. 382-386.
4. U. Inoue and S. Kawazu, "A Relational Database Machine for Very Large Information Retrieval Systems," in *Database Machines*, NATO ASI Series, Vol. F24, A.K. Sood and A.H. Qureshi, eds., Springer-Verlag, Berlin, 1986, pp. 183-201.
5. H. Takeda and T. Satoh, "An Accelerating Processor for Relational Operations," *Proc. Int'l Conf. Databases, Parallel Architectures, and Their Applications*, Mar. 1990, p. 559.
6. S. Shiokawa, Y. Obashi, and A. Nagoya, "DIPS-11/5E Series Mainframe," *Review of the ECL*, Vol. 35, No. 6, June 1987, pp. 633-641.
7. K. Itoh, R. Yazawa, and Y. Fukumura, "DIPS-V30 Development," *Review of the ECL*, Vol. 34, No. 5, May 1986, pp. 587-593.
8. S.G. Akl, *Parallel Sorting Algorithms*, Academic Press, New York, 1985.
9. M.W. Blasgen and K.P. Eswaran, "Storage and Access in Relational Data Bases," *IBM System J.*, Vol. 16, No. 4, 1977, pp. 363-377.
10. M. Kitsuregawa, M. Tanaka, and T. Moto-oka, "Application of Hash to Data Base Machine and Its Architecture," *New Generation Computing*, Vol. 1, No. 1, 1983, pp. 63-74.
11. T. Satoh et al., "Acceleration of Join Operations by a Relational Database Processor, RINDA," *Proc. Second Int'l Symp. Database Systems for Advanced Applications*, Apr. 1991, pp. 243-248.
12. D.R. McGregor, R.G. Thomson, and W.N. Dawson, "High Performance Hardware for Database Systems," in *Systems for Large Data Bases*, P.C. Lockemann and E.J. Neuhold, eds., North-Holland, Amsterdam, 1976, pp. 103-116.
13. V.Y. Lum, P.S.T. Yuen, and M. Dodd, "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files," *Comm. ACM*, Vol. 14, No. 4, Apr. 1971, pp. 228-239.
14. G.D. Knott, "Hashing Functions," *Computer J.*, Vol. 18, No. 3, Aug. 1975, pp. 265-287.
15. T. Satoh, H. Takeda, and N. Tsuda, "A Compact Multiway Merge Sorter Using VLSI Linear-Array Comparators," *Proc. Int'l Conf. Foundation Data Organization and Algorithms*, June 1989, pp. 223-227.
16. N. Tsuda, T. Satoh, and T. Kawada, "A Pipeline Sorting Chip," *Proc. IEEE Int'l Solid-State Circuits Conf.*, IEEE CS Press, Los Alamitos, Calif., 1987, pp. 270-271.
17. D. Bitton, D.J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems—A Systematic Approach," CTSR #526, Univ. of Wisconsin-

Madison, 1983.

18. D.J. DeWitt et al., "A Single User Evaluation of the GAMMA Database Machine," in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, eds., Kluwer Academic, 1988, pp. 370-386.



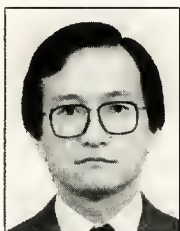
Ushio Inoue is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include information retrieval systems, database-management systems, real-time database systems, and database machines.

Inoue received a BE in electrical engineering from Nagoya University. He is a member of the IEEE Computer Society, the Institute of Electronics, Information and Communication Engineers of Japan, and the Information Processing Society of Japan.



Tetsuji Satoh is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include wafer-scale integration, parallel-processing architectures, hardware sorters, and database machines.

Satoh received a BE in electronic engineering from Yamanashi University. He is a member of the IEEE Computer Society, the Institute of Electronics, Information and Communication Engineers of Japan, and the Information Processing Society of Japan.



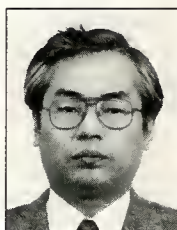
Haruo Hayami is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include mainframes and database machines.

Hayami received a BS and an MS in applied physics from Nagoya University. He is a member of the Information Processing Society of Japan.



Hideaki Takeda is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include parallel database algorithms, hardware sorters, and database machines.

Takeda received a BE and an ME in electrical engineering from Hokkaido University. He is a member of the IEEE Computer Society and the Information Processing Society of Japan.



Toshio Nakamura is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include compilers, database-management systems, and database machines.

Nakamura received a BE in electrical engineering from the Kyoto Institute of Technology. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.



Hideki Fukuoka is a senior research engineer at NTT Communications and Information Processing Laboratories. His research interests include mainframes and database machines.

Fukuoka received a BE in electronic engineering from the University of Osaka Prefecture. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.

Address questions regarding this article to Ushio Inoue, NTT Communications and Information Processing Laboratories, 1-2356 Take, Yokosuka-shi, Kanagawa 238-03, Japan; inoue@syrinx.ntt.jp.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate numbers on the Reader Service Card.

Low 165

Medium 166

High 167

1991 Index

IEEE Micro



This index covers all technical items — papers, correspondence, reviews, etc. — that appeared in *IEEE Micro* during 1991, and items from previous years that were commented upon or corrected in 1991.

The *Author Index* contains the primary entry for each item, listed under the first author's name, and cross-references from all coauthors. The *Subject Index* contains several entries for each item under appropriate subject headings, and subject cross-references.

It is always necessary to refer to the primary entry in the *Author Index* for the exact title, coauthors, and comments/corrections.

AUTHOR INDEX

A

- Abdelguerfi, M.**, and A. K. Sood. A fine grain architecture for relational database aggregation operations; *M-M Dec 91* 35-43
- Aschmann, Hans-Ruedi**, Niklaus Giger, Elisabeth Hoepli, Peter Janak, and Hubert Kirmann. Alphorn: A remote procedure call environment for fault-tolerant, heterogeneous, distributed systems; *M-M Oct 91* 16-19, 60-67
- Atkins, Mark**. Performance and the i860 microprocessor; *M-M Oct 91* 24-27, 72-78

B

- Balestra, Gabriella**, see Knaflitz, Marco, *M-M Oct 91* 12-15, 48-58
- Blasgen, Michael W.**, see Oehler, Richard R., *M-M Jun 91* 14-17, 56-62
- Brookwood, Nathan A.**, see Sachs, Howard G., *M-M Jun 91* 18-21, 74-80

C

- Caesar, Knut**, see Schmidt, Ulrich, *M-M Jun 91* 22-25, 88-94
- Chen, Ching-Ho**, see Li, Hua, *M-M Oct 91* 8-11, 44-47
- Chia, Wei-Kuo**, see Kuo, Yau-Hwang, *M-M Aug 91* 32-35, 86-92

D

- Di Stefano, Antonella**, Orazio Mirabella, and Fabio Presente. Implementing a DSP-based Petri-net simulation tool; *M-M Apr 91* 20-23, 56-64

F

- Farrell, James J., III**, see Seaborn, True, *M-M Feb 91* 5-9, 61

Faudemay, Pascal, and Mongia Mhiri. An associative accelerator for large data bases; *M-M Dec 91* 22-34

Fukuda, Akira, Kazuaki Murakami, and Shinji Tomita. Toward advanced parallel processing: Exploiting parallelism at task and instruction levels; *M-M Aug 91* 16-19, 50-61

Fukuoka, Hideki, see Inoue, Ushio, *M-M Dec 91* 61-70

Fulcher, John A. Fun and games and microcomputer interfacing; *M-M Feb 91* 18-21, 75-78

G

Gibson, Gary, see Popescu, Val, *M-M Jun 91* 10-13, 63-73

Giger, Niklaus, see Aschmann, Hans-Ruedi, *M-M Oct 91* 16-19, 60-67

H

- Haden, Kirtley B.**, see Russell, David W., *M-M Feb 91* 26-29
- Hall, Douglas V.** Adapting curriculum materials for different course sequences; *M-M Feb 91* 34-37, 82-83
- Hanson, Lee F.**, see Sachs, Howard G., *M-M Jun 91* 18-21, 74-80
- Hardie, Mark S.**, see Peyton Jones, Simon L., *M-M Feb 91* 38-41, 84-93
- Hayami, Haruo**, see Inoue, Ushio, *M-M Dec 91* 61-70
- Herman, Gary E.**, see Lee, Kuo Chu, *M-M Dec 91* 8-20
- Hickey, Takako Matoba**, see Lee, Kuo Chu, *M-M Dec 91* 8-20
- Hill, Mark D.**, and David A. Wood. Guest Editors' introduction: Hot chips II symposium (Special issue intro.); *M-M Jun 91* 8-9
- Hinata, Junichi**, see Yoshida, Toyohiko, *M-M Aug 91* 20-23, 62-72
- Hoepli, Elisabeth**, see Aschmann, Hans-Ruedi, *M-M Oct 91* 16-19, 60-67
- Hootman, Joe**, see Seaborn, True, *M-M Feb 91* 5-9, 61
- Hsiao, David K.** A parallel, scalable, and microprocessor-based database computer for performance gains and capacity growth; *M-M Dec 91* 44-60

I

- Inoue, Ushio**, Tetsuji Satoh, Haruo Hayami, Hideaki Takeda, Toshio Nakamura, and Hideki Fukuoka. RINDA: A relational database processor with hardware specialized for searching and sorting; *M-M Dec 91* 61-70

Isaman, David, see Popescu, Val, *M-M Jun 91* 10-13, 63-73

J

- Jaeger, Richard C.**, see Seaborn, True, *M-M Feb 91* 5-9, 61
- Janak, Peter**, see Aschmann, Hans-Ruedi, *M-M Oct 91* 16-19, 60-70
- Jeng, Guang-Huei**, see Kuo, Yau-Hwang, *M-M Aug 91* 32-35, 86-92

Jordan, Ramiro, *see* Pollard, L. Howard, *M-M Feb 91* 22-25, 78-79

K

- Kabemoto, Akira, and Hiroshi Yoshida. The architecture of the Sure System 2000 communications processor; *M-M Aug 91* 28-31, 73-78
- Kahaner, David K. Software Report—Optical computing activities; *M-M Feb 91* 53-56
- Kahaner, David K. Software Report—Computer growth in Taiwan; *M-M Jun 91* 39-41
- Kahaner, David K. Software Report—A glimpse of the future; *M-M Oct 91* 35, 79
- Kirrmann, Hubert. Micro World—When the computer was still personal; *M-M Feb 91* 42-44
- Kirrmann, Hubert. Micro World—Light at the end of the chunnel; *M-M Jun 91* 4-6
- Kirrmann, Hubert. Micro World—Europe on the way to the metric system; *M-M Aug 91* 36-39
- Kirrmann, Hubert, *see* Aschmann, Hans-Ruedi, *M-M Oct 91* 16-19, 60-67
- Klir, George J. Software Report—Advances in fuzzy theory and applications; *M-M Aug 91* 8-11
- Knaflitz, Marco, and Gabriella Balestra. Computer analysis of the myoelectric signal; *M-M Oct 91* 12-15, 48-58
- Kung, Ling-Yang, *see* Kuo, Yau-Hwang, *M-M Aug 91* 32-35, 86-92
- Kuo, Yau-Hwang, Ling-Yang Kung, Ching-Chung Tzeng, Guang-Huei Jeng, and Wei-Kuo Chia. KMDS: An expert system for integrated hardware/software design of microprocessor-based digital systems; *M-M Aug 91* 32-35, 86-92

L

- Lee, Kuo Chu, Takako Matoba Hickey, Victor W. Mak, and Gary E. Herman. VLSI accelerators for large database systems; *M-M Dec 91* 8-20
- Lehoczky, John P., *see* Sha, Lui, *M-M Jun 91* 30-33, 95-100
- Li, Hua, and Ching-Ho Chen. Simulating a function of visual peripheral processes with an analog VLSI network; *M-M Oct 91* 8-11, 44-47
- Lightner, Bruce, *see* Popescu, Val, *M-M Jun 91* 10-13, 63-73
- Louri, Ahmed. Three-dimensional optical architecture and data-parallel algorithms for massively parallel computing; *M-M Apr 91* 24-27, 65-82

M

- Mak, Victor W., *see* Lee, Kuo Chu, *M-M Dec 91* 8-20
- Mateosian, Richard. Review of 'Postscript Language Reference Manual, 2nd edn.' (Systems, A.; 1990); *M-M Apr 91* 28-29
- Mateosian, Richard. Review of 'Intel's Official Guide to 386 Computing' (Edelhart, M.; 1991); *M-M Jun 91* 50-51
- Mateosian, Richard. Review of 'Superscalar Microprocessor Design' (Johnson, M.); *M-M Jun 91* 51, 102
- Mateosian, Richard. Review of 'The Emperor's New Mind' (Penrose, R.; 1991); *M-M Aug 91* 42-43
- Mateosian, Richard. Review of 'Computer Dictionary'; *M-M Aug 91* 43
- Mateosian, Richard. Review of 'The Creative Mind—Myths and Mechanisms' (Boden, M. A.; 1990); *M-M Oct 91* 4-6
- McGhan, Harlan, *see* Sachs, Howard G., *M-M Jun 91* 18-21, 74-80
- Mhiri, Mongia, *see* Faudemay, Pascal, *M-M Dec 91* 22-34
- Mirabella, Orazio, *see* Di Stefano, Antonella, *M-M Apr 91* 20-23, 56-64
- Mizugaki, Shigeo, *see* Yoshida, Toyohiko, *M-M Aug 91* 20-23, 62-72
- Murakami, Kazuaki, *see* Fukuda, Akira, *M-M Aug 91* 16-19, 50-61
- Myers, Ware. The drive to the year 2000; *M-M Feb 91* 10-13, 68-74

N

- Nakamura, Toshio, *see* Inoue, Ushio, *M-M Dec 91* 61-70
- Nicoud, Jean-Daniel. Dedicated tools for microprocessor education; *M-M Feb 91* 14-17, 62-68

O

- Oehler, Richard R., and Michael W. Blasgen. IBM RISC System/6000: Architecture and performance; *M-M Jun 91* 14-17, 56-62

P

- Peterson, Craig, James Sutton, and Paul Wiley. iWarp: A 100-MOPS, LIW microprocessor for multicomputers; *M-M Jun 91* 26-29, 81-87
- Peyton Jones, Simon L., and Mark S. Hardie. A Futurebus interface from off-the-shelf parts; *M-M Feb 91* 38-41, 84-93
- Pollard, L. Howard, and Ramiro Jordan. An advanced educational microprocessor system; *M-M Feb 91* 22-25, 78-79
- Popescu, Val, Merle Schultz, John Spracklen, Gary Gibson, Bruce Lightner, and David Isaman. The metaflow architecture; *M-M Jun 91* 10-13, 63-73
- Presente, Fabio, *see* Di Stefano, Antonella, *M-M Apr 91* 20-23, 56-64
- Prete, Cosimo A. RST cache memory design for a tightly coupled multiprocessor system; *M-M Apr 91* 16-19, 40-52

R

- Rajkumar, Ragunathan, *see* Sha, Lui, *M-M Jun 91* 30-33, 95-100
- Roberts, Charles E. A RISC processor for embedded applications within an ASIC; *M-M Oct 91* 20-23, 68-72
- Rony, Peter R., *see* Seaborn, True, *M-M Feb 91* 5-9, 61
- Russell, David W., and Kirtley B. Haden. A configurable, virtual microprocessor system for instructional use in real-time, real-world studies; *M-M Feb 91* 26-29

S

- Sachs, Howard G., Harlan McGhan, Lee F. Hanson, and Nathan A. Brookwood. Design and implementation trade-offs in the Clipper C400 architecture; *M-M Jun 91* 18-21, 74-80
- Sakamura, Ken. Guest editor's introduction: Presenting the Far East special issue for 1991; *M-M Aug 91* 12-15
- Sakamura, Ken, *see* Takada, Hiroaki, *M-M Aug 91* 24-27, 78-85
- Satoh, Tetsuji, *see* Inoue, Ushio, *M-M Dec 91* 61-70
- Schmidt, Ulrich, and Knut Caesar. Datawave: A single-chip multiprocessor for video applications; *M-M Jun 91* 22-25, 88-94
- Schultz, Merle, *see* Popescu, Val, *M-M Jun 91* 10-13, 63-73
- Schultz, Thomas W. Peripheral hardware and a hands-on multitasking lab; *M-M Feb 91* 30-33, 80-82
- Seaborn, True, Peter R. Rony, Richard C. Jaeger, James J. Farrell III, and Joe Hootman. Looking back; *M-M Feb 91* 5-9, 61
- Sha, Lui, Ragunathan Rajkumar, and John P. Lehoczky. Real-time computing with IEEE Futurebus+; *M-M Jun 91* 30-33, 95-100
- Shimizu, Toru, *see* Yoshida, Toyohiko, *M-M Aug 91* 20-23, 62-72
- Slater, Michael. Micro View—The end of the 386 monopoly; *M-M Apr 91* 88, 87
- Slater, Michael. Micro View—Evolving architectures; *M-M Feb 91* 96-95
- Sood, A. K., *see* Abdelguerfi, M., *M-M Dec 91* 35-43
- Spracklen, John, *see* Popescu, Val, *M-M Jun 91* 10-13, 63-73
- Stern, Richard H. Micro Law—The paperback case; *M-M Apr 91* 30-33
- Stern, Richard H. Micro Law—(C):Software\Legal.hlp!; *M-M Jun 91* 42-46
- Stern, Richard H. Micro Law—The first chip-layout copying case; *M-M Aug 91* 3-6, 94
- Stern, Richard H. Micro Law—Fraud on the copyright office; *M-M Oct 91* 33-34
- Stern, Richard H. Micro Law—Database system copyrights; *M-M Dec 91* 77-79
- Stern, Richard M. Micro Law—The paperback case; *M-M Feb 91* 48-51
- Sutton, James, *see* Peterson, Craig, *M-M Jun 91* 26-29, 81-87

T

- Takada, Hiroaki**, and Ken Sakamura. ITRON-MP: An adaptive real-time kernel specification for shared-memory multiprocessor systems; *M-M Aug 91* 24-27, 78-85
Takeda, Hideaki, see Inoue, Ushio, *M-M Dec 91* 61-70
Tomita, Shinji, see Fukuda, Akira, *M-M Aug 91* 16-19, 50-61
Tzeng, Ching-Chung, see Kuo, Yau-Hwang, *M-M Aug 91* 32-35, 86-92

W

- Warren, Carl**. Micro Standards—There's a standard hiding out there; *M-M Feb 91* 45, 56
Warren, Carl. On the Edge—Evaluating shielded twisted-pair cable; *M-M Feb 91* 46-47
Warren, Carl. Micro Standards—Sorry, Captain Kirk; *M-M Apr 91* 34-35
Warren, Carl. On the Edge—Rate monotonic scheduling; *M-M Jun 91* 34-38, 102
Warren, Carl. Micro Standards—A bountiful return; *M-M Aug 91* 40-41
Warren, Carl. On the Edge—IEEE standard P1754: An open microprocessor architecture; *M-M Oct 91* 30-33
Warren, Carl. Micro Standards—Computing interconnections; *M-M Dec 91* 83-85
Wiley, Paul, see Peterson, Craig, *M-M Jun 91* 26-29, 81-87
Wood, David A., see Hill, Mark D., *M-M Jun 91* 8-9

Y

- Yoshida, Hiroshi**, see Kabemoto, Akira, *M-M Aug 91* 28-31, 73-78
Yoshida, Toyohiko, Toru Shimizu, Shigeo Mizugaki, and Junichi Hinata. The Gmicro/100 32-bit microprocessor; *M-M Aug 91* 20-23, 62-72

Z

- Zhang, Xiaodong**. System effects of interprocessor communication latency in multicomputers; *M-M Apr 91* 12-15, 52-55. *Correction*, *Jun 91* 6
Zhang, Xiaodong. Correction to 'System Effects of Interprocessor Communication Latency in Multicomputers' (Apr 91 12-15, 52-55); *M-M Jun 91* 6

SUBJECT INDEX

A

- Analog computers; cf.** Optical computing
Animation; cf. Visualization
Application-specific integrated circuits
 RISC processor for embedded applications within ASIC. *Roberts, Charles E.*, *M-M Oct 91* 20-23, 68-72
Array processing
 Datawave single-chip multiprocessor for video applications. *Schmidt, Ulrich*, +, *M-M Jun 91* 22-25, 88-94
Artificial intelligence; cf. Cognitive science
Associative memories
 associative caches and associative circuits as accelerators for large databases. *Faudemay, Pascal*, +, *M-M Dec 91* 22-34

B

- Bibliographies**
 3-D optical architecture. *Louri, Ahmed*, *M-M Apr 91* 24-27, 65-82
Biomedical signal analysis
 myoelectric signal analysis using computer systems and signal processing techniques. *Knaflitz, Marco*, +, *M-M Oct 91* 12-15, 48-58
Book reviews
 Computer Dictionary. *Mateosian, Richard*, *M-M Aug 91* 43

- Intel's Official Guide to 386 Computing (Edelhart, M.; 1991). *Mateosian, Richard*, *M-M Jun 91* 50-51
 Postscript Language Reference Manual, 2nd edn. (Systems, A.; 1990). *Mateosian, Richard*, *M-M Apr 91* 28-29
 Superscalar Microprocessor Design (Johnson, M.). *Mateosian, Richard*, *M-M Jun 91* 51, 102
 The Creative Mind—Myths and Mechanisms (Boden, M. A.; 1990). *Mateosian, Richard*, *M-M Oct 91* 4-6
 The Emperor's New Mind (Penrose, R.; 1991). *Mateosian, Richard*, *M-M Aug 91* 42-43

Brain; cf. Cognitive science

Buffer memories; cf. Cache memories

Bulk memories; cf. Mass memories

Business; cf. Computer industry

C

Cable shielding; cf. Wire communication cable shielding

Cache memories

- associative caches and associative circuits as accelerators for large databases. *Faudemay, Pascal*, +, *M-M Dec 91* 22-34
 RST cache memory design for tightly coupled Clipper-based multiprocessor system. *Prete, Cosimo A.*, *M-M Apr 91* 16-19, 40-52

Cellular logic

- Datawave single-chip multiprocessor for video applications. *Schmidt, Ulrich*, +, *M-M Jun 91* 22-25, 88-94

Circuit simulation

- KMDS expert system for integrated hardware/software design of microprocessor-based digital systems. *Kuo, Yau-Hwang*, +, *M-M Aug 91* 32-35, 86-92

Cognitive science

- book review: The Creative Mind—Myths and Mechanisms (Boden, M. A.; 1990). *Mateosian, Richard*, *M-M Oct 91* 4-6
 book review: The Emperor's New Mind (Penrose, R.; 1991). *Mateosian, Richard*, *M-M Aug 91* 42-43

Communication switching; cf. Message switching; Packet switching

Communication system performance; cf. Computer network performance

Compilers

- Metaflow architecture using deferred scheduling and register-renaming instruction shelf to manage out-of-order execution. *Popescu, Val*, +, *M-M Jun 91* 10-13, 63-73

Computer applications

- project management, site, track, and mobile computers used in construction of England-to-France tunnel (Micro World). *Kirmann, Hubert*, *M-M Jun 91* 4-6

Computer architecture

- 3-D optical architecture. *Louri, Ahmed*, *M-M Apr 91* 24-27, 65-82
 evolving architectures for microprocessors; overview. *Slater, Michael*, *M-M Feb 91* 96-95
 IBM RISC System/6000 architecture and performance; benchmark details. *Oehler, Richard R.*, +, *M-M Jun 91* 14-17, 56-62
 KRPP and DSNS superscalar architectures using task and instruction level parallelism. *Fukuda, Akira*, +, *M-M Aug 91* 16-19, 50-61
 Metaflow architecture using deferred scheduling and register-renaming instruction shelf to manage out-of-order execution. *Popescu, Val*, +, *M-M Jun 91* 10-13, 63-73
 redesign of Clipper C400 RISC architecture. *Sachs, Howard G.*, +, *M-M Jun 91* 18-21, 74-80

Computer buses; cf. Data buses

Computer control; cf. Digital control

Computer engineering education; cf. Computer science education

Computer graphics; cf. Visual languages; Visualization

Computer graphics languages; cf. Visual languages

Computer industry

- growth of computer industry in Taiwan (Software Report). *Kahaner, David K.*, *M-M Jun 91* 39-41

Computer interfaces

IEEE P1175/D11 draft standard on computing system tool interconnections overview (Micro Standards). *Warren, Carl, M-M Dec 91 83-85*

Lotus Development Corp. vs. Paperback Software International; court's failure to recognize functionality of standardization (Micro Law). *Stern, Richard M., M-M Feb 91 48-51*

real-time computing with IEEE Futurebus+. *Sha, Lui, +, M-M Jun 91 30-33, 95-100*

Computer interfaces; cf. Measurement-system data handling; Microcomputer interfaces

Computer language processors; cf. Compilers

Computer languages

Alphorn remote procedure call environment for fault-tolerant, heterogeneous, distributed systems. *Aschmann, Hans-Ruedi, +, M-M Oct 91 16-19, 60-67*

book review; Postscript Language Reference Manual, 2nd edn. (Systems, A.; 1990). *Mateosian, Richard, M-M Apr 91 28-29*

Computer languages; cf. Visual languages

Computer network performance

interprocessor communication performance evaluation of five types of hypercube and grid-topology multicomputers. *Zhang, Xiaodong, M-M Apr 91 12-15, 52-55+*

Computer networks; cf. Local area networks; Multiprocessing

Computer peripherals; cf. Microcomputer peripherals

Computer pipeline processing; cf. Pipeline processing

Computer science

book review; Computer Dictionary. *Mateosian, Richard, M-M Aug 91 43*

Computer science education

adapting curriculum materials for different sequences of microprocessing courses. *Hall, Douglas V., M-M Feb 91 34-37, 82-83*

advanced educational microprocessor system used as classroom demonstration tool and laboratory instrument. *Pollard, L. Howard, +, M-M Feb 91 22-25, 78-79*

microcomputer-interfacing laboratory projects. *Fulcher, John A., M-M Feb 91 18-21, 75-78*

microprocessor education curriculum incorporating logidules, CALM language, and Dauphin and Smaky microprocessor systems. *Nicoud, Jean-Daniel, M-M Feb 91 14-17, 62-68*

microprocessors in education (special issue). *M-M Feb 91 14-83*

teaching methods for peripheral hardware class and hands-on multitasking lab. *Schultz, Thomas W., M-M Feb 91 30-33, 80-82*

Computer vision; cf. Machine vision

Computers; cf. Database machines; Distributed computing; Microcomputers; Optical computing; Parallel processing; Virtual computers

Content-addressable memories; cf. Associative memories

Control engineering education

configurable, virtual microprocessor system to simulate and validate process plant designs in classroom. *Russell, David W., +, M-M Feb 91 26-29*

Control systems; cf. Digital control

Copyright protection

answering questions on copyright office forms correctly (Micro Law). *Stern, Richard H., M-M Oct 91 33-34*

database system copyrights (Micro Law). *Stern, Richard H., M-M Dec 91 77-79*

Copyright protection; cf. Software protection

Custom integrated circuits; cf. Application-specific integrated circuits

D

Data acquisition; cf. Measurement-system data handling

Data buses

IEEE P996 draft standard for AT-bus; development (Micro Standards). *Warren, Carl, M-M Feb 91 45, 56*

Data communication; cf. Distributed computing; Local area networks; Measurement-system data handling; Message switching; Multiprocessing, interconnection

Data processing; cf. Database systems

Data structures

associative caches and associative circuits as accelerators for large databases. *Faudemay, Pascal, +, M-M Dec 91 22-34*

Database machines

associative caches and associative circuits as accelerators for large databases. *Faudemay, Pascal, +, M-M Dec 91 22-34*

database machines (special issue). *M-M Dec 91 6-76*

multibackend database supercomputer interconnected by LAN; architecture and performance. *Hsiao, David K., M-M Dec 91 44-60*

RINDA relational database processor with hardware specialized for searching and sorting. *Inoue, Ushio, +, M-M Dec 91 61-70*

Database management systems; cf. Database systems, searching

Database systems

database system copyrights (Micro Law). *Stern, Richard H., M-M Dec 91 77-79*

Database systems; cf. Distributed database systems

Database systems, relational

fine grain architecture for relational database aggregation. *Abdelguerfi, M., +, M-M Dec 91 35-43*

RINDA relational database processor with hardware specialized for searching and sorting. *Inoue, Ushio, +, M-M Dec 91 61-70*

VLSI accelerators for improving large database system performance. *Lee, Kuo Chu, +, M-M Dec 91 8-20*

Database systems, searching

RINDA relational database processor with hardware specialized for searching and sorting. *Inoue, Ushio, +, M-M Dec 91 61-70*

VLSI accelerators for improving large database system performance. *Lee, Kuo Chu, +, M-M Dec 91 8-20*

Design automation; cf. Circuit simulation; Design automation software

Design automation software

shielded twisted-pair cable characteristics evaluation using Math CAD (On the Edge). *Warren, Carl, M-M Feb 91 46-47*

Digital control

configurable, virtual microprocessor system to simulate and validate process plant designs in classroom. *Russell, David W., +, M-M Feb 91 26-29*

Digital integrated circuits; cf. Very-large-scale integration

Digital signal processors; cf. Microprocessors

Disk recording; cf. Optical memories

Distributed computing

iWarp microprocessor supporting message-passing and systolic communications for multicomputers. *Peterson, Craig, +, M-M Jun 91 26-29, 81-87*

Distributed computing; cf. Distributed database systems; Multiprocessing

Distributed database systems

Alphorn remote procedure call environment for fault-tolerant, heterogeneous, distributed systems. *Aschmann, Hans-Ruedi, +, M-M Oct 91 16-19, 60-67*

multibackend database supercomputer interconnected by LAN; architecture and performance. *Hsiao, David K., M-M Dec 91 44-60*

E**Education**

microprocessors in education (special issue). *M-M Feb 91 14-83*

Education; cf. Computer science education

Educational technology; cf. Computer science education; Visualization

Electrical engineering education; cf. Control engineering education

EMG (electromyography); cf. Muscles, EMG

England; cf. United Kingdom

Europe

use of English units in European hardware and software systems (Micro World). *Kirrmann, Hubert, M-M Aug 91 36-39*

Expert systems

KMDS expert system for integrated hardware/software design of microprocessor-based digital systems. *Kuo, Yau-Hwang, + , M-M Aug 91 32-35, 86-92*

F

Fault tolerance; cf. Redundant systems

Forecasting; cf. Technology forecasting

France

project management, site, track, and mobile computers used in construction of England-to-France tunnel (Micro World). *Kirrmann, Hubert, M-M Jun 91 4-6*

Fuzzy set theory

fuzzy set theory applications and advances in Japan (Software Report). *Klir, George J., M-M Aug 91 8-11*

H

History

microcomputer developments during past ten years (Micro World). *Kirrmann, Hubert, M-M Feb 91 42-44*

tenth anniversary retrospective. *Seaborn, True, + , M-M Feb 91 5-9, 61*

Holographic memories

3-D optical architecture. *Louri, Ahmed, M-M Apr 91 24-27, 65-82*

Human factors; cf. Cognitive science

I

IEEE Micro

tenth anniversary retrospective. *Seaborn, True, + , M-M Feb 91 5-9, 61*

IEEE standards

developments in several draft standards (Micro Standards). *Warren, Carl, M-M Aug 91 40-41*

IEEE open microprocessor architecture standard P1754 (On the Edge). *Warren, Carl, M-M Oct 91 30-33*

IEEE P1175/D11 draft standard on computing system tool interconnections overview (Micro Standards). *Warren, Carl, M-M Dec 91 83-85*

IEEE P996 draft standard for AT-bus; development (Micro Standards). *Warren, Carl, M-M Feb 91 45, 56*

real-time computing with IEEE Futurebus+. *Sha, Lui, + , M-M Jun 91 30-33, 95-100*

Information systems; cf. Database systems

Instrumentation; cf. Measurement

Integrated circuits; cf. Application-specific integrated circuits; Microprocessors; Very-large-scale integration

J

Japan

Far East (special issue). *M-M Aug 91 12-31*

fuzzy set theory applications and advances in Japan (Software Report). *Klir, George J., M-M Aug 91 8-11*

K

Knowledge-based systems; cf. Expert systems

L

LAN; cf. Local area networks

Languages; cf. Computer languages

Learning systems; cf. Neural networks

Legal factors

Brooktree Corp. vs. Advanced Micro Devices, Inc.; issues arising from first chip-layout copying case (Micro Law). *Stern, Richard H., M-M Aug 91 3-6, 94*

Legal factors; cf. Copyright protection; Software protection

Local area networks

multibackend database supercomputer interconnected by LAN; architecture and performance. *Hsiao, David K., M-M Dec 91 44-60*

M

Machine vision

2D analog VLSI network simulating function of human visual, peripheral processes. *Li, Hua, + , M-M Oct 91 8-11, 44-47*

Management; cf. Project management

Mass memories

trends in hardware and applications of microcomputers for next ten years. *Myers, Ware, M-M Feb 91 10-13, 68-74*

Measurement

use of English units in European hardware and software systems (Micro World). *Kirrmann, Hubert, M-M Aug 91 36-39*

Measurement-system data handling

Futurebus interface design using off-the-shelf parts for graph reduction in parallel (GRIP) system. *Peyton Jones, Simon L., + , M-M Feb 91 38-41, 84-93*

Measurement-systems data handling; cf. Data buses

Memories; cf. Associative memories; Cache memories; Holographic memories; Mass memories; Microcomputer memories; Optical memories

Memory management; cf. Protocols, memory

Mental models; cf. Cognitive science

Message switching

interprocessor communication performance evaluation of five types of hypercube and grid-topology multicomputers. *Zhang, Xiaodong, M-M Apr 91 12-15, 52-55†*

Microcomputer applications

trends in hardware and applications of microcomputers for next ten years. *Myers, Ware, M-M Feb 91 10-13, 68-74*

Microcomputer interfaces

microcomputer-interfacing laboratory projects. *Fulcher, John A., M-M Feb 91 18-21, 75-78*

Microcomputer memories

trends in hardware and applications of microcomputers for next ten years. *Myers, Ware, M-M Feb 91 10-13, 68-74*

Microcomputer networks; cf. Distributed computing; Multiprocessing

Microcomputer performance

IBM RISC System/6000 architecture and performance; benchmark details. *Oehler, Richard R., + , M-M Jun 91 14-17, 56-62*
redesign of Clipper C400 RISC architecture. *Sachs, Howard G., + , M-M Jun 91 18-21, 74-80*

Microcomputer peripherals

teaching methods for peripheral hardware class and hands-on multitasking lab. *Schultz, Thomas W., M-M Feb 91 30-33, 80-82*

Microcomputer peripherals; cf. Microcomputer interfaces

Microcomputer software

3-D optical architecture. *Louri, Ahmed, M-M Apr 91 24-27, 65-82*

Microcomputers

microcomputer developments during past ten years (Micro World). *Kirrmann, Hubert, M-M Feb 91 42-44*

Microcomputers; cf. Microprocessors**Microprocessor applications; cf. Specific topic****Microprocessors**

- book review; Intel's Official Guide to 386 Computing (Edelhart, M., 1991). *Mateosian, Richard*, *M-M Jun 91* 50-51
- Datawave single-chip multiprocessor for video applications. *Schmidt, Ulrich*, +, *M-M Jun 91* 22-25, 88-94
- DSP-based Petri-net simulation tool. *Di Stefano, Antonella*, +, *M-M Apr 91* 20-23, 56-64
- evolving architectures for microprocessors; overview. *Slater, Michael*, *M-M Feb 91* 96-95
- Gmicro/100 32-b microprocessor based on TRON specifications. *Yoshida, Toyohiko*, +, *M-M Aug 91* 20-23, 62-72
- Hot Chips II: Sweetening the pot (special issue). *M-M Jun 91* 8-29
- i860 microprocessor design and performance. *Atkins, Mark*, *M-M Oct 91* 24-27, 72-78
- IBM RISC System/6000 architecture and performance; benchmark details. *Oehler, Richard R.*, +, *M-M Jun 91* 14-17, 56-62
- IEEE open microprocessor architecture standard P1754 (On the Edge). *Warren, Carl*, *M-M Oct 91* 30-33
- iWarp microprocessor supporting message-passing and systolic communications for multicomputers. *Peterson, Craig*, +, *M-M Jun 91* 26-29, 81-87
- Metaflow architecture using deferred scheduling and register-renaming instruction shelf to manage out-of-order execution. *Popescu, Val*, +, *M-M Jun 91* 10-13, 63-73
- redesign of Clipper C400 RISC architecture. *Sachs, Howard G.*, +, *M-M Jun 91* 18-21, 74-80
- RISC processor for embedded applications within ASIC. *Roberts, Charles E.*, *M-M Oct 91* 20-23, 68-72
- trends in hardware and applications of microcomputers for next ten years. *Myers, Ware*, *M-M Feb 91* 10-13, 68-74

Microprocessors; cf. Microcomputers; Multiprocessing...**Modeling; cf. Petri nets; Simulation****Multiprocessing**

- 3-D optical architecture. *Louri, Ahmed*, *M-M Apr 91* 24-27, 65-82
- DSP-based Petri-net simulation tool. *Di Stefano, Antonella*, +, *M-M Apr 91* 20-23, 56-64
- Futurebus interface design using off-the-shelf parts for graph reduction in parallel (GRIP) system. *Peyton Jones, Simon L.*, +, *M-M Feb 91* 38-41, 84-93
- ITRON-MP adaptive real-time kernel specification for shared-memory multiprocessor systems. *Takada, Hiroaki*, +, *M-M Aug 91* 24-27, 78-85
- KRPP and DSNS superscalar architectures using task and instruction level parallelism. *Fukuda, Akira*, +, *M-M Aug 91* 16-19, 50-61
- RST cache memory design for tightly coupled Clipper-based multiprocessor system. *Prete, Cosimo A.*, *M-M Apr 91* 16-19, 40-52
- Sure System 2000 fault-tolerant computer using hardware and software local redundancies. *Kabemoto, Akira*, +, *M-M Aug 91* 28-31, 73-78

Multiprocessing; cf. Array processing; Distributed computing**Multiprocessing, interconnection**

- interprocessor communication performance evaluation of five types of hypercube and grid-topology multicomputers. *Zhang, Xiaodong*, *M-M Apr 91* 12-15, 52-55†

Multiprocessing, interconnection; cf. Local area networks**Multiprocessors**

- ITRON-MP adaptive real-time kernel specification for shared-memory multiprocessor systems. *Takada, Hiroaki*, +, *M-M Aug 91* 24-27, 78-85

Multitasking

- teaching methods for peripheral hardware class and hands-on multitasking lab. *Schultz, Thomas W.*, *M-M Feb 91* 30-33, 80-82

Muscles, EMG

- myoelectric signal analysis using computer systems and signal processing techniques. *Knaflitz, Marco*, +, *M-M Oct 91* 12-15, 48-58

N**Networks; cf. Multiprocessing, interconnection; Neural networks; Petri nets****Neural networks**

- optical computing overview; relation to neural networks (Software Report). *Kahaner, David K.*, *M-M Feb 91* 53-56

O**Optical computing**

- optical computing overview; relation to neural networks (Software Report). *Kahaner, David K.*, *M-M Feb 91* 53-56
- trends in hardware and applications of microcomputers for next ten years. *Myers, Ware*, *M-M Feb 91* 10-13, 68-74

Optical memories

- 3-D optical architecture. *Louri, Ahmed*, *M-M Apr 91* 24-27, 65-82

P**Packet switching**

- Futurebus interface design using off-the-shelf parts for graph reduction in parallel (GRIP) system. *Peyton Jones, Simon L.*, +, *M-M Feb 91* 38-41, 84-93

Parallel processing

- book review; Superscalar Microprocessor Design (Johnson, M.). *Mateosian, Richard*, *M-M Jun 91* 51, 102
- IBM RISC System/6000 architecture and performance; benchmark details. *Oehler, Richard R.*, +, *M-M Jun 91* 14-17, 56-62
- Metaflow architecture using deferred scheduling and register-renaming instruction shelf to manage out-of-order execution. *Popescu, Val*, +, *M-M Jun 91* 10-13, 63-73
- rate monotonic scheduling algorithm applied to programming real-time systems (On the Edge). *Warren, Carl*, *M-M Jun 91* 34-38, 102

Parallel processing; cf. Multiprocessing; Pipeline processing**Parallel processing, interconnection; cf. Multiprocessing, interconnection****Patents; cf. Software protection****Personal computers; cf. Microcomputers****Petri nets**

- DSP-based Petri-net simulation tool. *Di Stefano, Antonella*, +, *M-M Apr 91* 20-23, 56-64

Pipeline processing

- fine grain architecture for relational database aggregation. *Abdelguerfi, M.*, +, *M-M Dec 91* 35-43
- redesign of Clipper C400 RISC architecture. *Sachs, Howard G.*, +, *M-M Jun 91* 18-21, 74-80

Project management

- project management, site, track, and mobile computers used in construction of England-to-France tunnel (Micro World). *Kirrmann, Hubert*, *M-M Jun 91* 4-6

Protocols

- Futurebus interface design using off-the-shelf parts for graph reduction in parallel (GRIP) system. *Peyton Jones, Simon L.*, +, *M-M Feb 91* 38-41, 84-93

Protocols, memory

- RST cache memory design for tightly coupled Clipper-based multiprocessor system. *Prete, Cosimo A.*, *M-M Apr 91* 16-19, 40-52

Publishing; cf. Copyright protection**R****Real-time systems**

- ITRON-MP adaptive real-time kernel specification for shared-memory multiprocessor systems. *Takada, Hiroaki*, +, *M-M Aug 91* 24-27, 78-85
- rate monotonic scheduling algorithm applied to programming real-time systems (On the Edge). *Warren, Carl*, *M-M Jun 91* 34-38, 102

real-time computing with IEEE Futurebus+. *Sha, Lui, +, M-M Jun 91* 30-33, 95-100

Redundant systems

Sure System 2000 fault-tolerant computer using hardware and software local redundancies. *Kabemoto, Akira, +, M-M Aug 91* 28-31, 73-78

Routing; cf. Multiprocessing, interconnection

S

Scheduling

Metaflow architecture using deferred scheduling and register-renaming instruction shelf to manage out-of-order execution. *Popescu, Val, +, M-M Jun 91* 10-13, 63-73

rate monotonic scheduling algorithm applied to programming real-time systems (On the Edge). *Warren, Carl, M-M Jun 91* 34-38, 102

Search methods; cf. Database systems, searching

Semicustom integrated circuits; cf. Application-specific integrated circuits

Set theory; cf. Fuzzy set theory

Shielding; cf. Wire communication cable shielding

Signal analysis; cf. Biomedical signal analysis

Signal processing; cf. Video signal processing

Simulation

configurable, virtual microprocessor system to simulate and validate process plant designs in classroom. *Russell, David W., +, M-M Feb 91* 26-29

DSP-based Petri-net simulation tool. *Di Stefano, Antonella, +, M-M Apr 91* 20-23, 56-64

Simulation; cf. Circuit simulation

Software; cf. Computer languages; Design automation software;

Microcomputer software; Multitasking

Software, utility programs; cf. Computer interfaces

Software design/development; cf. Data structures

Software education; cf. Computer science education

Software protection

Ashton-Tate Corp. vs. Fox Software, Inc; applicability of patent and copyright laws to software (Micro Law). *Stern, Richard H., M-M Jun 91* 42-46

Lotus Development Corp. vs. Paperback Software International; court's failure to recognize functionality of standardization (Micro Law). *Stern, Richard M., M-M Feb 91* 48-51

Lotus Development Corp. vs. Paperback Software International; effects of courts decision on software industry (Micro Law). *Stern, Richard H., M-M Apr 91* 30-33

Software standards

Lotus Development Corp. vs. Paperback Software International; court's failure to recognize functionality of standardization (Micro Law). *Stern, Richard M., M-M Feb 91* 48-51

Lotus Development Corp. vs. Paperback Software International; effects of courts decision on software industry (Micro Law). *Stern, Richard H., M-M Apr 91* 30-33

Sorting/merging; cf. Database systems, relational

Special issues/sections

database machines. *M-M Dec 91* 6-76

Far East. *M-M Aug 91* 12-31

Hot Chips II: Sweetening the pot. *M-M Jun 91* 8-29

microprocessors in education. *M-M Feb 91* 14-83

Standards

difficulties arising from diversity of methods for obtaining benchmarks (Micro Standards). *Warren, Carl, M-M Apr 91* 34-35

Futurebus interface design using off-the-shelf parts for graph reduction in parallel (GRIP) system. *Peyton Jones, Simon L., +, M-M Feb 91* 38-41, 84-93

Standards; cf. IEEE standards; Software standards

T

Taiwan

growth of computer industry in Taiwan (Software Report). *Kahaner, David K., M-M Jun 91* 39-41

Technology forecasting

trends in hardware and applications of microcomputers for next ten years. *Myers, Ware, M-M Feb 91* 10-13, 68-74

Terminology

book review; Computer Dictionary. *Mateosian, Richard, M-M Aug 91* 43

TV; cf. Video signal processing

Twisted-pair cables

shielded twisted-pair cable characteristics evaluation using Math CAD (On the Edge). *Warren, Carl, M-M Feb 91* 46-47

U

Uncertain systems; cf. Fuzzy set theory

United Kingdom

project management, site, track, and mobile computers used in construction of England-to-France tunnel (Micro World). *Kirrmann, Hubert, M-M Jun 91* 4-6

V

Very-large-scale integration

2D analog VLSI network simulating function of human visual, peripheral processes. *Li, Hua, +, M-M Oct 91* 8-11, 44-47

Gmicro/100 32-b microprocessor based on TRON specifications. *Yoshida, Toyohiko, +, M-M Aug 91* 20-23, 62-72

VLSI accelerators for improving large database system performance. *Lee, Kuo Chu, +, M-M Dec 91* 8-20

Video signal processing

Datawave single-chip multiprocessor for video applications. *Schmidt, Ulrich, +, M-M Jun 91* 22-25, 88-94

Virtual computers

configurable, virtual microprocessor system to simulate and validate process plant designs in classroom. *Russell, David W., +, M-M Feb 91* 26-29

Vision systems (nonbiological); cf. Machine vision

Visual languages

book review; Postscript Language Reference Manual, 2nd edn. (Systems, A.; 1990). *Mateosian, Richard, M-M Apr 91* 28-29

Visualization

configurable, virtual microprocessor system to simulate and validate process plant designs in classroom. *Russell, David W., +, M-M Feb 91* 26-29

VLSI; cf. Very-large-scale integration

W

Wire communication cable shielding

shielded twisted-pair cable characteristics evaluation using Math CAD (On the Edge). *Warren, Carl, M-M Feb 91* 46-47



Richard H. Stern

Law Offices of

Richard H. Stern

1300 19th Street NW,

Suite 400

Washington, DC 20036

Database system copyrights

Databases, or more generally digital information, differ in important ways from the kind of information traditionally protected under copyright law. Protecting databases by copyright law impinges on different interests of plaintiffs and defendants than does protecting books, pictures, and songs—all traditional subjects of copyright law. A sensible legal policy may therefore call for different trade-offs among competing demands of owners, competitors, and users to maximize the benefits to the public—or minimize the public harm—resulting from the operation of the legal system.

Differences from traditional subjects

Commentators¹ have pointed out ways in which databases differ from the traditional subject matter of copyright law.

- Digital information is often more easily and cheaply copied, making appropriation of such information easier and more profitable in the short run than appropriation of traditional works.
- Copyright law regards as significant the distinctions among literary works (words), audiovisual works (pictures, graphics, changing sequences of imagery), and musical works (sound). But these distinctions blur for digital information. It is all bits or bytes stored on tape, disk, or CD-ROM, no matter how you slice it. Why treat use of some bits differently?
- Books can be read by eye. Pictures are also visually perceived. Music or other sound can simply be listened to and apprehended by ear. Most users come equipped with these hardware (or wetware) devices. Databases cannot be used without appropriate special-purpose hardware and software.

- Finally, digital information is more easily modified into another form. You cannot readily turn the *Mona Lisa* into something else. But any time you access a database you can rearrange the downloaded material into almost any desired format. Users can and do readily create new things with old material, making new works that may be eligible for copyright protection. The *Mona Lisa* is in the public domain. So is Beethoven's *Eroica*. With a database and appropriate hardware and software, a user can create a new *Eroic-Mona* pastiche.

Another aspect of database systems, as they are now developing, defies even comparison with the traditional subject matter of copyright. This is the issue of what's the data and what's the program. I do not mean that you cannot inspect bits in a CD-ROM and figure out which bits comprise data and which bits comprise instructions that manipulate the data. You may be able to do this, if someone has provided a debugger or disassembler or reverse compiler; otherwise, probably not. But so what?

Who owns the data?

The problem I mean is that a database user gets results by using a legally protectable computer program to manipulate data that is largely unprotectable in itself. This is because the data is old or simply was not created or owned by the proprietor of the database. Sweat equity in collecting and storing a mass of factual data, such as names and addresses of people who have telephone numbers in a town, is not legally protected under copyright law except for original aspects of selection and arrangement, if any. (The Supreme Court recently so held in a decision that a telephone company held no copyright on its white pages directory, because there was no au-

thorial originality in the content.²⁾

The economic value of the database—and of any product you can create from it—comes from both the data and the software for accessing and organizing it. Yet the resulting product does not visibly contain the software. It just contains the data, in a differently manipulated format. The result is like a compiled program that does not include any “runtime modules.” (Runtime modules are units of code written for a compiler package and incorporated into a compiled code as part of compilation of source code.) Traditional copyright doctrine would suggest there is no computer program in the result and thus nothing legally protected. But that result may not make the best sense from a business standpoint or in terms of encouraging desired forms of enterprise. Opinions will differ.

You might take that example further, in terms of user interfaces. One of the things that makes a database easy to use is a well-designed user interface. In a sense, the interface contributes to the ease of making new products from a database and the quality level they reach, but the new product does not visibly contain the interface itself. Some might consider the resulting legal implications a cause for regret. Others will applaud.

Blurred lines of authorship

Generally speaking, copyright law protects only results that embody authorial originality—some kind of creative spark by a person claiming to have done the creating. This legal principle impinges on databases in two ways. First, the proprietor of the database system, including whatever software is involved, cannot claim to be the creator of what some other person (a user) brings into existence by using the system. Building a soldering iron does not make you the creator of a breadboard whose components someone else solders together.

Second, there may be a question about whether even the person who

uses the database system to produce something is an author of an original work. The person may not contribute enough, either. This may be a situation in which authorship falls into the bit bucket.

The same kind of question has been raised for hypertext linkages for database entries. A hypertext linkage is a facility like a footnote. If this text (for example, the preceding sentence) were on your screen because it belonged to a database that you were using (who knows, *IEEE Micro* on line for hypertext may be just around the corner), you could mouse click on the word “hypertext” (or otherwise enter it). A window would appear, showing a definition of hypertext. Or you could click on the word “footnote” and a window would pop up, explaining what a footnote is. That would occur, however, only if someone went over this text and created a set of hypertext linkages. The linkages would invoke routines that make the program controlling display of this text jump to an entry elsewhere in the database system, to stored text describing hypertext, footnotes, or whatever.

Creating such hypertext linkages is said to be a form of scholarship. Doubt has been expressed, however, that such scholarship results in any copyrightable work of authorship. If that means that no effective reward system will encourage creation of hypertext linkages, perhaps there is a social problem.

On the other hand, creation of some hypertext linkages for my text might offend me. That, too, could cause a social problem. Suppose that someone links this text to a database dictionary entry to exemplify the term “GIGO.” Perhaps, I would not want my work linked in that manner. Should authors have control over hypertext linkages to or from their work? Under traditional copyright law principles, I can prevent my work from being reproduced without permission (subject to fair-use limitations on my rights). Hypertext linkages may dilute that right. Is there a

copyright infringement that I can suppress by law when someone makes a hypertext linkage? Probably not. There is no unauthorized reproduction of my work (once it has lawfully been placed into the database), public performance of it, or other recognized category of copyright infringement.

Those are some of the problems we can anticipate when the courts try to apply copyright law to database systems. It is the same kind of problem you run into when you try to use a set of legal categories developed for one purpose as a mechanism for regulating something else that somebody says ought to be regarded as the same thing and therefore regulated in the same way. What is ever the same as anything else that went before? That has been a problem since the Greeks looked at rivers or tried to address convergence toward a limit. (See box on next page.)

Sometimes this legal process works, because the two things are enough the same for the purpose at hand, or are sufficiently the same considering how much (or little) money you want to spend to resolve the issue. Other times, the process works dismally, because the new subject matter is so vastly different from the old that the legal results do not pass the well-known subjective tests. (These tests have various names, such as the “straight face” test or the “barf” test. The question is whether the judgments that courts reach using the legal theory in question are too ridiculous.)

It would be premature to pass judgment now on whether applying copyright law to the use of database systems will pass or fail these tests. The real-world chickens haven't come home to roost, yet.

References

1. P. Samuelson, “Digital Media and the Changing Face of Intellectual Property Law,” *Rutgers Computer & Technology J.*, Vol. 16, 1990, p. 323.

2. Feist Publications, Inc. v. Rural Telephone Service Co., 111 S. Ct. 1282 (1991).

When are two things the same?

Heracleitus was one of the first to address this issue when he asked if one ever steps twice into the same river. Zeno then came up with a misguided way of addressing a series of the form $t = (s/v)(1 + 0.1 + 0.01 + \dots)$. He believed there was no answer, that is, no knowable value for t , simply because the series was infinite.

Zeno argued that Achilles, running 10 times as fast as a tortoise creeps, could never catch up to it. Each time Achilles closed a gap ds , the tortoise advanced another increment of $0.1 ds$, and thus would forever stay ahead. (If Zeno believed that, he would have been willing to believe a great many other things.)

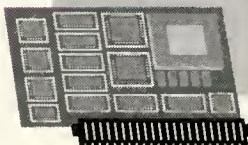
This is much like the logic of copyright law and indeed legal logic in general. Which parameters you designate as independent variables determine the outcome. If you decide to talk about $F(g)$, the plaintiff wins; if you decide to talk about $G(f)$, the defendant wins. It's like integrating by parts, where $duv = u dv + v du$. What you decide to designate u and what you decide to designate v (or dv) determines whether you will solve the problem or just make it worse.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card

Low 177 Medium 178 High 179

On the Edge



Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-3311

x. 6-0669

warren@ssdunix.mdc.com

SBus: an open bus architecture

[In the third part of our series on tools, Rudolf Usselmann discusses the SBus and its progress towards IEEE standardization.]

I invite readers to send information on a tool or method that solves problems, for consideration in future columns.—C.W.]

Rudolf Usselmann
Sparc International

Sun Microsystems originally developed the SBus for the Sparcstation 1 as an inexpensive, high-performance system interconnection bus. "The original goal of the SBus was to provide an expansion bus for free, as our cost target for the Sparcstation 1 was zero dollars," said Jim Ludemann of Antares Microsystems. Ludemann was one of the original codesigners of Sparcstation 1 and SBus.

"This meant we had to design a bus flexible enough for I/O, but fast enough to connect main memory with the cache," Ludemann said. "The SBus was literally the only bus in Sparcstation 1. We knew that in future system implementations the SBus would be used only for I/O, as system design considerations require a custom memory bus for each new system. Furthermore, SBus was designed from the start to be implemented with CMOS gate arrays,

giving us low cost, high performance, and low power usage."

As we can see in many implementations of SBus, the company followed its plans precisely. Even clone manufacturers followed these steps and implemented SBus in their systems. A good example is the Toshiba Laptop, which has one SBus slot for extensions.

Features

SBus offers a wide variety of advantages for system and peripheral developers that other buses cannot offer as a package. Individually, its features are easily achieved and are incorporated in other buses. As a combined package, they offer a very good solution for modular I/O subsystems extensions. (See Table 1 on page 81 for detailed signal definitions.)

Low device count. Various companies offer one-chip solutions for SBus developers. These chips integrate the complete SBus interface on one IC. Peripheral developers have an easy task connecting standard I/O devices (serial I/O, SCSI, and LAN controllers) to single chips and don't have to worry about correct SBus implementation. System developers can purchase interface chips that allow communication from high-speed CPU interconnection buses (for example, MBus) to SBus, thus simplifying the task of system development dramatically.

Low-power CMOS implementation. SBus is 100 percent compatible

with CMOS technology, which eliminates the need for high current drivers and large power supplies, as well as problems with heat dissipation.

Flexible high performance. The original implementation of SBus allows data transfer rates up to 80 Mbytes/s. After the company released SBus to public domain, users formed an SBus working group and started working on an extension.

The SBus public committee developed an extension for SBus to allow 64-bit operations. This meant defining new transfer types and introducing modes without breaking current implementations. It was not a simple task.

The extension of SBus to 64 bits increases performance to more than 160 Mbytes/s. This will help I/O performance keep up with ever-increasing CPU performance. In the past, many I/O devices could not connect to an I/O bus because of their need for high throughput. The extensions will allow

these devices to connect to SBus. However, SBus sizing does not limit one to a certain transfer rate. SBus dynamic bus sizing and a wide variety of cycle types can fit many needs.

Open Boot. Open Boot firmware supports the SBus and assists software drivers. Each SBus board has a small ROM containing identification information and optional boot and diagnostic drivers. The identification information includes such generic items as the device name, register addresses, and interrupt levels and may include device-specific items such as the resolution of a display device.

This ROM information is stored as a computer program written in F Code, a byte-coded version of the Forth programming language. F Code is independent of the CPU's instruction set architecture, so the SBus device's F Code ROM can be used without change on systems with different CPU types.

The Open Boot firmware residing on

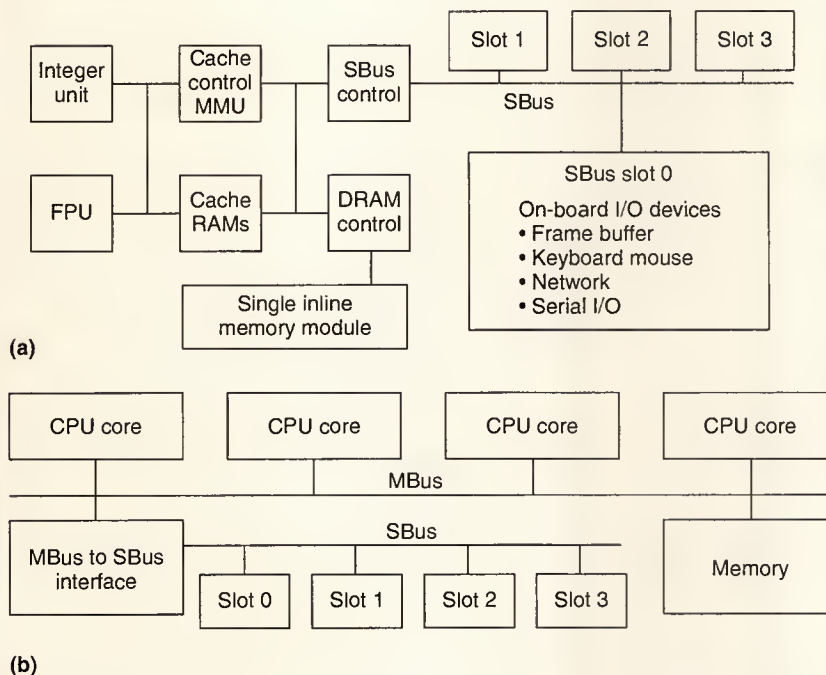


Figure 1. Low-cost (a) and high-performance (b) systems.

Contacts

SBus study group

e-mail: sbus_sg@sparc.com
(Mail goes to 150 members)

Wayne Fischer
Chair, SBus study group
Force Computers
3165 Winchester Blvd.
Campbell, CA 95008
phone: (408) 370-6300
fax: (408) 374-1146
e-mail: uunet!force-clwfischer

SBus study group alias maintenance and requests

e-mail: sbus_sgreq@sparc.com

SBus Specifications

Hamilton Avnet Electronics
(800) 442-6458

Sun's Interoperability Center

Yatin Trivedi
2550 Garcia Ave., PAL1-106
Mountain View, CA 94043
(415) 336-1812
e-mail: trivedi@sun.com

the CPU board contains an interpreter for the F Code language. Interpretation of the F Code ROM creates an entry for the device in the firmware's device tree, a hierarchical data structure describing the system configuration. The operating system software inspects this device tree to determine which devices are present in the system and the individual characteristics of those devices.

If the F Code ROM contains diagnostic and/or boot drivers, those drivers may be used by the firmware for such tasks as testing the device hardware, loading the operating system from that device, or displaying start-up messages using the device.

Open Boot's F Code scheme is not limited to SBus. F Code is being considered for use with Futurebus+ and

Table 1. Signal definitions for SBus standard 32-bit and extended 64-bit modes. All signals are distributed via a high-density 96-pin connector, which also includes GND, +5V, and $\pm 12V$ supplies.

Signal	I/O	Driven by	Description
PA(27:00) [D(59:32)]*	I	Controller	Physical address
Sel_	I	Controller	Slave select (one per slave)
D(31:0) [D(31:0)]*	I/O	Masters/slave	Data
Siz(2:0) [D(62:60)]*	I/O	Masters	Transfer size
Rd [D(63)]*	I/O	Masters	Transfer direction
AS_	I	Controller	Address strobe
Ack(2:0)_	I/O	Slaves/controller	Transfer acknowledgment
LErr_	I/O	Slaves	Late data error
BR_	O	Masters	Bus request (one per master)
BG_	I	Controller	Bus grant (one per master)
Clk	I	Controller	SBus clock
Reset_	I	Controller	Reset
IntRes(7:1)_	O	Slaves	Interrupt request (open drain)
DtaPar	I/O	Master/slaves	Data parity (optional)

In an extended mode, the signals below are time multiplexed

Signal	Description
D(31)	Extended transfer type
D(30:28)	Extended transfer size (2:0)
D(27)	Extended transfer read
D(26:25)	Extended transfer atomic (1:0)
D(24:0)	Extended transfer reserved (24:0)

Supported sizes

Signal	Description	Description
ETSiz(2:0)		Extended (64-bit) mode
Siz(2:0)	Standard (32-bit) mode	
000	Word (4-byte) transfer	Reserved
001	Byte transfer	Reserved
010	Half-word (2-byte) transfer	Reserved
011	Extended Transfer	8 bytes
100	Four-word burst (16 bytes)	16 bytes
101	Eight-word burst (32 bytes)	32 bytes
110	Sixteen-word burst (64 bytes)	64 bytes
111	Two-word burst (8 bytes)	128 bytes

Acknowledgment encoding

Ack(2:0)_	Function
111	Idle/Wait
110	Error acknowledgment
101	Byte (data) acknowledgment
100	Rerun acknowledgment
011	Word (data) acknowledgment
010	Double-word (data) acknowledgment
001	Half-word (data) acknowledgment
000	Reserved

Atomic cycle types

ETAtomic(1:0)	Description
00	Normal bus cycle (non-atomic bus cycle)
01	First bus cycle of an atomic transaction
10	Intermediate bus cycle of an atomic transaction
11	Last cycle of an atomic transaction

* Signals in square brackets are for 64-bit mode. They are shared with the regular signals when that mode is enabled.

_ Signals with an underscore mark indicate low-active signals.

VME-D, and its design applies to other buses as well. Open Boot firmware is the subject of an IEEE standardization effort under project number P1275.

Current status

Sun Microsystems has placed SBus in the public domain and helped to establish a Public Specification Com-

mittee to continue developing the SBus specification. This committee is working on extensions to SBus and is addressing the undefined and unclear

aspects of its specification.

Ongoing project

Because of the wide acceptance of the SBus standard, Wayne Fischer of Force Computers asked the IEEE Computer Society and the Bus Architecture Standards Committee to open a Project Authorization Request (PAR) to establish SBus as an IEEE standard. In June 1991 IEEE's Bus Architecture Standards Committee agreed to sponsor a study group. At the June meeting of the SBus Public Spec Committee, Fischer suggested this group merge into the SBus study group. The committee unanimously approved. At publication, this group has met three times, most recently on December 3-4. The next meeting is scheduled for January 22-23, 1992, in Salt Lake City.

Interoperability Center

The Interoperability Center offers SBus card developers help in designing hardware and software and assists with porting SBus subsystems to the company's different platforms. It is equipped with most hardware and software tools a designer might need, including logic analyzers, oscilloscopes, and most of the popular schematic editors available on the market. The

center has a well-experienced staff to assist designers with problems.

Compatibility

Today vendors produce more than 400 different SBus boards. Thus, compatibility is becoming an increasingly important issue to vendors and end users. Sparc International introduced, in summer 1991, a program for compatibility testing of SBus boards on Sparc-compliant platforms. This service verifies the compatibility of SBus boards and accommodating software across a wide variety of platforms.

In a joint effort with VME Labs, Sparc International plans to expand this program to include full system-independent hardware and software verification by mid 1992. VME Labs plans to perform hardware verification; Sparc International plans to test device drivers and other accommodating software. Currently the lack of a device driver standard makes testing impossible. However, Sparc International and its members are developing a uniform standard for the next Unix release SVR4.

Summary

Considering the typical lifetime of an I/O bus, SBus is in its infancy and has a lot of room to grow. If it becomes a

standard, its growth will be guaranteed in a controlled manner, not in chaos, like other buses. Furthermore, the standardization of SBus will ensure its continued support and expandability.

Acknowledgments

I thank Jim J. Ludemann of Antares Microsystems, Mitch Bradley of Sun Microsystems, and Wayne Fischer of Force Computers and the SBus study group for their help with this article.

Rudolf Usselmann, a senior architect at Sparc International in Menlo Park, Calif., develops extensions to the Sparc architecture and related products such as the SBus and MBus and performs compliance testing on Sparc processors and peripherals.

Usselmann holds a Diploma in computer science and electrical engineering from the University of Hamburg.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card

Low 192 Medium 193 High 194

CALL FOR PAPERS!

SPECIAL ISSUE, IEEE DESIGN & TEST OF COMPUTERS ON DESIGN AND TEST OF MEGABIT MEMORIES

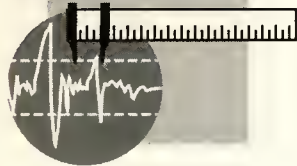
IEEE D&T will devote its March 1993 special issue to a wide range of topics on the design and test of high-density and high-speed memories. The memory design process that in the past had been the domain of circuit and process development experts now requires expertise in various other fields to implement testability features and device architectures for high-speed access using pipelining, interleaving, and so on.

DEADLINES

Submit six copies of a 20-page, double-spaced, typewritten paper **before April 1, 1992**. You will be notified by September 1, 1992, if your paper is accepted. Final version of your paper must arrive no later than November 1, 1992.

Send submissions to Manuel d'Abreu, editor-in-chief, GE Corporate R&D, 1 River Rd., Bldg. KW-C308, Schenectady, NY 12301; (518) 387-7097; dabreu@crd.ge.com. Direct any questions to either the editor-in-chief or guest editors Pinaki Mazumder or John P. Hayes, Dept. of EE&CS, University of Michigan, Ann Arbor, MI 48109-2122; (313) 763-2107 or (313) 763-0386; mazum@indus.eecs.umich.edu or jhayes@eecs.umich.edu.

Micro Standards



Computing interconnections

The emphasis on networking puts a new burden on system designers and software engineers: the interchange of data in a seamless fashion. Many methods have been considered and have proven workable at least on a one-to-one basis. For example, early on *Byte* magazine attempted to provide a common format for information interchange using the Kansas City Standard for audiotape recording of computer data and programs. The system worked in the context of the machines and users at that time.

Publishers of the now-defunct *Interface Age* magazine, of which incidentally I served as editor-in-chief, saw in 1977 the need for ensuring that common data was used in the many different processor types and operating systems. The solution was the IPIS (International Platform Interchange System). This approach consisted of a software process that ensured that data either conformed to a specific format or could be translated to that format with some manipulation on a host or target system. The problems with IPIS were that no ground swell requirement for interchange existed and operating systems and languages were still being defined. Essentially we had a solution that was out of context with the need at the time. We had come up with an interesting addition to computer science, but it didn't catch on.

Robust systems

Today, manufacturers deliver platforms with robust and well-defined operating systems and environments. It isn't unusual to find a large network consisting of Macintoshes, IBM PCs, Sun and Hewlett-Packard workstations, and large mainframes. In some cases the ability to interchange data is common due to the operating

environment. Unix machines, for example, provide not only a common file transport mechanism (File Transport Protocol—FTP or Network File System—NFS) but also file and data formats that are similar if not exactly the same.

Not all systems run Unix, and not all Unix boxes are implemented in the same way. Consequently, not all data can be guaranteed to translate to another platform the same way. A Unix-to-VAX/VMS interchange not only requires translating syntax and methods of naming files but the basic file structures as well.

To this end, work has been going on to develop a common interconnection tool that ensures the easy and obvious way of moving and sharing data.

The interconnection tool

The IEEE effort is P1175/D11 (draft 11), A Standard Reference Model for Computing System Tool Interconnections, prepared by the IEEE Computer Society's Task Force on Professional Computing Tools. The balloting committee has approved P1175, which awaits final approval by the IEEE Standards Board this month. For copies of P1175, telephone the IEEE Standards office in Piscataway, New Jersey, at (800) 678-IEEE; the fax number there is (908) 981-9667. For answers to questions about the draft standard, contact P1175 Chair Robert M. Poston, Programming Environments, Inc., 4043 State Highway 33, Tinton Falls, NJ 07753; telephone (908) 918-0110 or fax (908) 918-0113.

The P1175 document describes a robust solution to resolving interconnection problems called the Semantic Transfer Language (STL). The standard describes STL as parsable, easy to read and understand by programmers, easy to write, and easy to convert to a compact transfer form. The

Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-3311

x. 6-0669

warren@ssdunx.mdc.com

notion of the STL describes items in the form of English sentences that contain one subject, a verb or verb phase, and only one relation or attribute to the subject. To maintain commonality with existing systems and languages, STL uses a modified version of the Backus-Naur Form (BNF) for syntax description. For example, ::= means "is defined as," and white spaces or tabs mean "is followed by." Similarly, STL relates items by concepts and actions or functions of the concepts, such as those listed in the box.

Interestingly, the overall concept of P1175 and STL is based on communications concepts overlaid onto system and software methods. Within the software constraints the issues of concern are actions, transformation of state, control and data; events, control, timing, and synchronization of actions; information and data, the primary object of software actions; logic, logical restrictions (conditions) on actions; and finally states, the context for and evolution of actions.

Standard not limited

The architects of P1175 haven't limited the standard by looking only at a narrow range of hardware and software concepts. Rather, they began at the uppermost part—the user hierarchy, the organization. They thought that if information can't be easily and readily shared by an organization, it isn't useful. Moreover, the organization establishes the operation context and defines the platform(s) that will be used. For example, McDonnell Douglas primarily uses Macintosh IIcx systems and VAX mainframes, while supporting subcontractors using IBM PCs and a variety of other platforms.

The next issue addressed by the working group was tool-to-platform interconnections. This so-called architectural context is guided by the organization context. When the organization is closed, the process is simple, and platform-to-platform interconnections work easily. However when the

mix is great either within or external to an organization, the problem becomes greater, and tools must be available for interconnecting platforms. The most logical way to solve the problem is to use networks that range from local area networks (LANs) to wide area networks (WANs).

Once the organization and architectural issues and interconnections were worked out, the committee tackled the problem of interconnections among tools, the transfer context. How does data element A on platform A probably get to platform Z over the network and remain a useful entity? Enter STL. The Semantic Transfer Language provides the mechanism to assist in the translation—not a trivial concept.

Not a lonely effort

The IEEE effort isn't one that was pursued without support. Indeed, many members of the task team represent organizations and other standards groups that are interested in information interchange and interconnections. One notable effort that has been tak-

ing place resulted from NASA's Space Station Freedom. The space station will be using a processing system to collect and download sensor data to earth stations. This system brought about the development of a Data Naming Standard and a translation tool for ensuring that all the associated databases function in a common way. Thus the schema (the structure of the database) of one database can effectively communicate with another database that has a different purpose.

With this in mind and using communications concepts, the company developed the Interdiscipline Systems Definition Encyclopedia (ISDE). This tool allows us to define database schemas in a common way and provides a translation mechanism so that any database can be made to appear to conform to a common format. ISDE is implemented as a database for the Macintosh using 4th Dimension Database Manager software from Acius Corp. The software checks schemas against the McDonnell Douglas Data and Object Standards (DAOS) naming standard for conformity.

Although McDonnell Douglas designed ISDE to solve a specific problem associated with the space station, it arranged ISDE to follow the same rules as P1175. However, the approach is one of a relational database rather than a semantic transfer language. Both methods have merit and should be combined. Therefore, I'm recommending to the P1175 chair that the ISDE model be considered as an addendum to the basic P1175 work. Combining the STL with a relational database tool will result in a common tool definition that will ensure accurate information interchange regardless of platform or implementation. However, although I will seek a Project Authorization Request for this work, P1175 first faces a two-year trial-use period (beginning shortly) before an addendum can be considered.

For additional information about the Data and Object Naming Standard

STL concepts

- Action
- Collection
- Condition
- ConnectionPath
- Constant
- DataItem
- DataKey
- DataPart
- DataRole
- DataStore
- DataType
- DataView
- EventItem
- EventType
- GraphicSymbol
- Object
- S_Packet
- State
- StateTransition

(DAOS), contact C.R. Easton, McDonnell Douglas Space Systems Co., Space Station Div., 5301 Bolsa Ave., M/S A95-J845-17/6; Huntington Beach, CA 92647; telephone (714) 896-4551; fax (714) 896-5034.

For more information about ISDE, contact Lee Neitzel, CTA Corp., 18333 Egret Bay Blvd., Suite 310, Houston, TX 77058; telephone (713) 333-2436; fax (713) 333-2493.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 180 Medium 181 High 182

1951-1991 **IEEE Computer Society Press Books**

MULTIPLE-VALUED LOGIC IN VLSI DESIGN edited by Jon T. Butler

This book provides a historical perspective on MVL, focuses on various technologies for implementing multiple-valued VLSI circuits, delves into applications of MVL in diverse technologies, and discusses device physics, logic characteristics, and small and medium-scale circuit experiences. The text contains 12 papers divided into four categories—Introduction, Multiple-Valued Logic Technology, Special Implementations, and Multiple-Valued Logic Tools and Techniques.

128 PAGES. JULY 1991. SOFTBOUND.
ISBN 0-8186-2127-3.
CATALOG NO. 2127 \$35.00
MEMBERS \$28.00

**Call toll-free
to order
1-800-CS-BOOKS
(in CA (714) 821-8380)**



Joe Hootman

University of North Dakota

Send announcements of new microcomputer and microprocessor products to
Managing Editor, IEEE Micro, PO Box 3014, Los Alamitos, CA 90720-1264

Workstations

Entry- and mid-level workstations

Users who have outgrown their personal computers but aren't ready to move up to high-end workstations may want to consider Sun's IPX and EPX Sparcstations.

The 16-Mbyte IPX (expandable to 64 Mbytes) is a mid-level workstation designed for a variety of applications, including complex, computer-aided software development, financial analysis, electronic publishing, and network file/database access. It achieves 28.5 MIPS and 4.2 Mflops with a rating of 24.2 Specmarks. GX accelerated graphics speed window response time and allow users to manipulate multiple windows and complex objects in near-real time. IPX works as a single unit or configures as a file server for small work groups.

The ELC yields 21 MIPS and 3 Mflops with 20.1 Specmarks. It offers multiple windows, a high-resolution display, fast response times, built-in networking, and 8 Mbytes of standard memory (expandable to 16 Mbytes). It also configures as a file server. ELC integrates all of the workstation components into the back of a monochrome display. *Sun Microsystems*; \$11,995 (IPX), \$4,995 (ELC).

Reader Service No. 10

25-MHz SBus Sparc

Compstation 25 is an SBus-compatible, Sparc-compliant definition 1.1 workstation that runs on Sparc/OS 1.1.1. It yields 15.8 MIPS and 1.75 Mflops with 10.25 Specmarks. The system features a high-resolution, 19-in. color monitor and 8 Mbytes of RAM (expandable to 64 Mbytes). It supports Sun View, Open Windows, and Motif software. An SBus add-on card allows users simultaneous access to Unix and DOS environments. *Tatung Science and Technology*; \$7,995.

Reader Service No. 11

16-Mbyte Sparc CPU

The power and software capability of a Sun workstation running SunOS is available to the embedded systems market in a 16-Mbyte Sparc CPU. The Sparc CPU-1E/16 is a VME single-board computer that offers enough on-board memory to run standard SunOS in a single-slot environment without a memory expansion board. It achieves 12.5 MIPS and 1.4 Mflops with 8.4 Specmarks. *Force Computers*; \$7,995.

Reader Service No. 12

Computer boards and cards

CMOS 68000 single-board computer

Designed for use in remote or battery-powered portable applications, the

MPL-4079 board consumes 120mA with the CPU running at 8 MHz. Six 32-pin JEDEC sockets can be equipped with up to 512 Kbytes of PROM and 256 Kbytes of battery-backed RAM on a 25-sq in. CMOS RAM.

The device also has two RS-232 serial ports, a battery-backed clock calendar, two 16-bit timers, and a four-channel, 8-bit, A/D converter. The board's G-64 bus interface expands I/O capabilities. It operates from -40° to +85° C. Software for the MPL-4079 can be developed on DOS and downloaded to the board's memory using Crosscode C and PC bridge cross development software packages. *Gespac*; \$595 (100s).

Reader Service No. 13

PCs can run Macintosh software

DOS users can install the Andor One add-in board with associated software to run Apple Macintosh software. The chip's circuitry allows a DOS machine's 3.5-in. drive to directly read and write Macintosh disks. The full-length PC card installs in any PC, from the earliest PC-XTs through 486-based machines. Andor One has terminate-and-stay-resident software that takes 60 Kbytes of RAM. The package also includes Word-for-word/Mac, a DOS-Mac translator.

According to the company, Andor One runs twice as fast as a Macintosh Classic. It is compatible with DOS-standard peripherals such as mouse, keyboard, hard disk, 3.5-in. disk drives, and monitors. An Apple Talk-compatible RS-422 connector allows it to network with Apple Laserwriters, Apple Local Talk, Phonenet, and other networking devices. The board requires Mac-plus ROM and Macintosh Systems and Finder software. *Hydra Systems*; \$995.

Reader Service No. 14

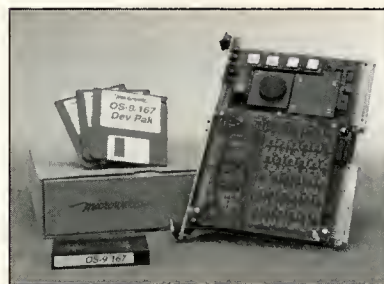
Enhanced OS-9 system

OS-9/MVME167 Development Pak is an optimized version of the OS-9 real-

time operating system. Its manufacturer says it takes full advantage of the power of Motorola's 32-bit M68040 processor while providing support for the advanced on-board serial, SCSI, and Ethernet hardware.

The package includes a number of OS-9 device drivers for I/O peripherals included on the MVME167 board family, including SCSI drivers for the NCR 53C710 controller. Other drivers support the on-board, real-time clock and the CD-2401 serial I/O controller. An Internet Support Package supports Ethernet. Also included are a set of resident development tools for application development, including K&R C compiler, macro assembler and linker, user state debugger, μMACS full-screen editor, and shell command interpreter. The system also comes in a modified Run-Time Pak that excludes device drivers and development tools. *Microware Systems*; \$3,000 (Development Pak), \$1,500 (Run-Time Pak).

Reader Service No. 15



Microware's OS-9 Development Pak

Classic runs twice as fast

The Classic Performer boosts the speed and performance of the Macintosh Classic by as much as 96 percent, its manufacturer says. The accelerator board is a 68000 processor that runs at 16 MHz, twice the clock speed of the Classic's built-in processor. It has a 64-Kbyte SRAM cache circuit and 25-ns PAL chips. The manufacturer says it speeds math calculations by up to 75 percent and accelerates the SCSI port by 15 percent. An optional math

coprocessor boosts math-intensive programs by as much as 5,000 percent. It supports Macintosh systems above 6.0.7. *Harris Laboratories*; \$299.95 (Classic Performer) \$149.95 (68881 math coprocessor).

Reader Service No. 16

Communications software

Data compression for micro-to-mainframe conversion

Version 2 of Compress is an advanced bit compression/decompression utility that processes text and binary files and offers ASCII/EBCDIC conversion tables. The manufacturer says it reduces size and transfer time of text files by 50 to 80 percent.

Compress is compatible with most micro-to-mainframe products with file transfer capabilities. Version 2 also supports OS/2 and Macintosh workstations. It is available in VM/CMS, DOS/VSE CICS, and MVS (TSO and CICS) versions. *Telepartner International*; from \$6,000. Free upgrades for customers on maintenance.

Reader Service No. 17

Network management

The two products in the Network Control Series offer managers the ability to control their enterprisewide networks at central and local levels. Cornerstone Agent allows subnet managers to monitor traffic on the network, filter data, generate statistics, and decode protocols. It identifies potential LAN problems or interfaces with Foundation Manager to allow a central administrator to troubleshoot any LAN on the network. Foundation Manager has in-depth analysis and management capabilities including statistical analysis, network characterization, simulation, auto-baselining, protocol analysis, and network visual mapping.

Both products are based on a graphically programmable user interface and built-in start-up programs. The series supports MIB I, MIB II, and Remote

Network Monitoring MIB. It also supports Token Ring source routing and IP routing. *Pro Tools*; \$8,995 (*Foundation Manager*), \$1,295 (*Cornerstone Agent*). Upgrades for \$500.

Reader Service No. 18

VAX-to-IBM connection

VAX users can connect directly to IBM's LU6.2/Token Ring with EZ Bridge. This software allows VAX users on PCs or VT terminals to log on to a mainframe as a 3270 user and access mainframe applications such as CICS and TSO. VAX users can also print DOS-format files on their local printers and have peer-to-peer communications with LU6.2 systems.

The EZ Bridge family includes three products. The SNA/3270 connects VAX to mainframe by emulating an IBM 3174 or 3274 cluster controller, 3278/9 terminals, and 3287 printers. The SNA/LU6.2 implements IBM's LU6.2 protocol that provides peer-to-peer communications capabilities on an SNA/Token Ring network. EZ Bridge OLTP allows users to access and update data in different locations. *Systems Strategies*; \$3,000 to \$15,000, based on VAX size.

Reader Service No. 19

Communications hardware and chips

Multimedia communications chip

Fax, data, voice, and caller ID capabilities are combined on an LSI chip, the Fax Vodem (YTM403). The chip enables users to send and receive graphics, data, and voice messages on a single line and incorporate caller ID if desired. It supplies voice quality with resolution up to 12 bits at 9,600 bps and dynamic range up to 72 decibels.

A built-in voice record-and-playback capability provides a 12-to-4 bit, three-to-one compression and decompression ratio. An internal analog-to-digital/digital-to-analog converter supports voice-mail capability. A caller ID func-

tion allows receivers to identify the sender before answering the phone or receiving a communication. The chip comes in 64-pin SDIP or 64-pin QFP versions. *Yamaha Systems Technology*; \$40 (1,000s).

Reader Service No. 20

10 Base T through AUI port on Ethernet

The Model 177 transceiver attaches to Ethernet equipment to upgrade LAN coaxial Ethernet adapters and wiring networks to the 10 Base T wiring standard. It uses unshielded, twisted-pair wiring and connects to Ethernet through an AUI port or via an Ethernet transceiver cable.

The unit has four status LEDs indicating link, collision, receive, and transmit. Power derives from the AUI port, making an external source unnecessary. The 10 Base T port uses an RJ45 connector. *Telebyte Technology*; \$119, discounts for quantities.

Reader Service No. 21



Telebyte's Model 177 transceiver

Chips and components

3V CMOS EPROMs

The 27LV256 and the 27LV512 are 3V CMOS EPROMs designed for battery-powered applications in which 5V devices are no longer preferred. The

manufacturer says the 3V EPROMs provide 200-ns access times.

The 27LV256 and 27LV512 are available in speeds of 200, 250, and 300 ns. Both devices come in plastic dip, PLCC, and SOIC packages. *Microchip Technology*; 27LV256 from \$4.73, 27LV512 from \$8.03 (1,000s).

Reader Service No. 22

Hardware-based fuzzy-logic controller

The NLX230 is a single-chip fuzzy-logic inference engine with on-chip rule memory. The manufacturer says it processes 30 to 40 times faster than comparable software-based or hardware/software hybrid fuzzy-logic control solutions. The manufacturer recommends it as a replacement for conventional PID controllers, sequencers, state machines, and intelligent timers.

The controller is housed in a 40-pin package. It supports eight inputs, eight outputs, and a resident base of 64 fuzzy-logic rules. The manufacturer also offers an applications development system that includes the chip and associated circuitry on a PC-compatible board, controlling software, and documentation. *Neura Logic*; \$4 (production quantities), \$395 (development system).

Reader Service No. 23

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Advertiser/Product Index

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northwest Region: D. Rodney Brooks; Tel: (415) 905-0260; Fax: (415) 896-1512.

Eastern Region: Georgette Boone; Tel: (415) 905-0260; Fax: (415) 896-1512.

Southwest Region: D. Rodney Brooks; Tel: (415) 905-0260; Fax: (415) 896-1512.

Recruitment and Classified Advertising: D. Rodney Brooks; Tel: (415) 905-0260; Fax (415) 896-1512.

Director of Sales: Randall L. Stickrod, 544 Second St., Suite 200, San Francisco, CA 94107; Tel: (415) 905-0260; Fax: (415) 896-1512.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.



Have you heard about our...

Technical Committee on Microprocessors and Microcomputers

*For information on this, or any of our more than
30 technical committees, contact:*

IEEE COMPUTER SOCIETY
Membership/Circulation Dept.
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1264
(714) 821-4010

RS # Page

Force Computers	12	86
Gespac	13	87
Harris Laboratories	16	87
Hydra Systems	14	87
IEEE CS Membership	—	21
Microchip Technology	22	88
Microware Systems	15	87
Neura Logix	23	88
Pro Tools	18	88
Sun Microsystems	10	86
Systems Strategies	19	88
Tatung Science and Technology	11	86
Telebyte Technology	21	88
Telepartner International	17	87
Yamaha Systems Technology	20	88

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office; to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204

(requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

1951-1991

40 YEARS OF SERVICE



IEEE COMPUTER SOCIETY

A member society of the
Institute of Electrical and Electronics Engineers, Inc.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes six magazines and five research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

EXECUTIVE COMMITTEE

President: Duncan H. Lawrie*
University of Illinois
Dept. of Computer Science
1304 W. Springfield
Urbana, IL 61801
(217) 333-3373

President-Elect: Bruce D. Shriver*
Past President: Helen M. Wood*

VP, Standards: Paul L. Borrell (1st VP)*
VP, Press Activities: Barry W. Johnson (2nd VP)*
VP, Conferences and Tutorials: Laurel V. Kaleda†
VP, Education: Raymond E. Miller†
VP, Membership Activities: Ronald O. Williams†
VP, Publications: Ronald G. Hoelzeman†
VP, Technical Activities: Mario R. Barbacci*

Secretary: James H. Aylor*
Treasurer: Joseph Boykin†
Division V Director: Edward A. Parrish, Jr.†
Division VIII Director: Helen M. Wood*
Executive Director: T. Michael Elliott†

*voting member of the Board of Governors
†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 1991:

P. Bruce Berra, Michael Evangelist,
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,
Joseph E. Urban, Thomas W. Williams

Term Expiring 1992:

James H. Aylor, Alicia I. Ellis, Tadao Ichikawa,
C.V. Ramamoorthy, Sallie V. Sheppard,
Harold Stone, Akihiko Yamada

Term Expiring 1993:

Fiorenza Albert-Howard, Jon T. Butler, Michael C. Mulder,
Yale N. Patt, Anneliese von Mayrhauser,
Benjamin W. Wah, Ronald Waxman

Next Board Meeting

February 28, 1992, 8:30 a.m.
Cathedral Hill Hotel, San Francisco, CA

SENIOR STAFF

Executive Director: T. Michael Elliott
Publisher: H. True Seaborn
Director, Conferences and Tutorials: Anne Marie Kelly
Director, Finance and Information Services: Tod S. Heisler
Director, Board and Administrative Services: Violet S. Doan
Assistant to the Executive Director: Sandra K. Pfau

COMPUTER SOCIETY OFFICES

Headquarters Office

1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone: (202) 371-0101
Fax: (202) 728-9614

Publications Office

10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information:
(714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

European Office

13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asian Office

Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 3408-3118
Fax: 81 (3) 3408-3553



IEEE OFFICERS

President: Eric E. Sumner
President Elect: Merrill W. Buckley, Jr.
Past President: Carleton A. Bayless
Secretary: Hugh Rudnick
Treasurer: Theodore W. Hissey, Jr.

VP, Educational Activities: Richard S. Nichols
VP, Professional Activities: Michael J. Whitelaw
VP, Publication Activities: J. Thomas Cain
VP, Regional Activities: Robert T. H. Alden
VP, Technical Activities: Fernando Aldana